Filipe Rafael Oliveira Torrado

Licenciado em Engenharia Electrotécnica e de Computadores

Collaborative Study Web Platform

Dissertação para obtenção do Grau de Mestre em

Engenharia Electrotécnica e de Computadores

Orientador: Luís Manuel Camarinha-Matos, Professor Catedrático, Universidade Nova de Lisboa

Júri

Presidente: Prof. Doutor João Almeida das Rosas, Universidade Nova de Lisboa
Arguente: Prof. Doutora Patrícia Alexandra Pires Macedo, Instituto Politécnico de Setúbal
Vogal: Prof. Doutor Luís Manuel Camarinha-Matos, Universidade Nova de Lisboa

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Maio, 2016

Collaborative Study Web Platform

Este documento foi gerado utilizando o processador (pdf)LaTeX, com base no template "unlthesis" [1] desenvolvido no Dep. Informática da FCT-NOVA [2].
[1] https://github.com/joaomlourenco/unlthesis [2] http://www.di.fct.unl.pt

Acknowledgements

I would like to thank Professor Luís Camarinha-Matos, adviser of the dissertation, for keeping my goals clear and realistic, focusing on what really mattered on the project. To all members of NEEC, the students group of the Electrical and Computer Engineering Department, from where the idea of the project originated. To João Aires, that provided some help on making a few of the page interface elements, my special thanks.

Also to the team at Eggy, where I have been working on web application development. Special thanks to Pedro Reis Ferreira, work and college colleague and good friend. Last, but not least, I would like to acknowledge and thank my parents for the support throughout the course. To all my friends that have been supporting me, specially to my girlfriend Raquel Almeida, which has been a great friend and supportive of my efforts.

Abstract

Online learning is gaining importance in Universities, in student communities, and companies. Learning Management Systems (LMS) are the prevailing software solutions for education and e-learning. These systems are evolving from simple containers of courses information, syllabus, and documents or files, to offer more intricate features, associated with social and collaboration-oriented software. Aside from LMS, students often use multiple web applications for studying and doing projects. LMS may not offer a workspace where users can organize and share information, which is segmented on several online tools. Some solutions may even entirely restrict creation of content by students.

With this in mind, a system was designed and implemented with the goal of providing an alternative or complementary solution to other LMS. In general, educational institutions deploy these systems, with restricted access within their peers. On the contrary, the proposed approach provides a set of collaboration and content organization tools. It is a web application provided under a software as a service (SaaS) model, to which potentially anyone can access and register.

The platform is organized into groups, which hold content elements and place users together. Each group member has his/her set of roles inside the group, defining corresponding permissions, which are enforced by an access control system. Permissions are set in respect to users, groups and content.

An emphasis is given on providing a way to assess or rate users through both their actions and content creation, hypothesizing this as a factor for user engagement and trust. Users, through several feedback elements such as voting and commenting, evaluate content. A presentation is done of the studied rating calculation methods and simulation of several of these methods.

The resulting web platform sets a basis to explore different approaches for content creation and sharing collaboratively. The use cases in the system are analysed and discussed, considering this system as a foundation for a web application focused on collaborative and group-based study.

Keywords: Web application, Learning management systems, Active learning, Collaboration, Content sharing, Access control, Rating

Resumo

A aprendizagem online tem ganho importância junto de Universidades, comunidades de alunos e também empresas. Os sistemas de gestão de aprendizagem (LMS) são as soluções de *software* predominantemente utilizadas para educação e *e-learning*. Estes sistemas têm vindo a progredir, de simples "depósitos" de informação sobre o curso, programas e documentos ou ficheiros, para passarem a oferecer características mais complexas associadas a *software* social ou orientado para a colaboração. Além dos LMS, os estudantes usam frequentemente múltiplas aplicações *web* para estudar e para realizar projectos. Os LMS podem não oferecer um espaço de trabalho onde os utilizadores possam organizar e partilhar informação, que se encontra segmentada por diversas ferramentas online. Algumas soluções restringem completamente a criação de conteúdo pelos estudantes.

Tendo isto em conta, um sistema foi concebido e implementado com o propósito de ser uma alternativa ou ser complementar a outros LMS. Tipicamente, as instituições de ensino utilizam tais sistemas com acesso restrito aos seus associados. Pelo contrário, a abordagem proposta fornece um conjunto de ferramentas colaborativas e de organização de conteúdos. Fornecida num modelo de *software as a service* (SaaS), onde potencialmente qualquer pessoa pode registar-se e ter acesso.

A plataforma é organizada em grupos, que dispõem de conteúdos e agrupam os utilizadores. Cada membro tem um conjunto de papéis no grupo, definindo as suas permissões, as quais são garantidas pelo sistema de controlo de acessos. As permissões são definidas ao nível dos utilizadores, grupos e conteúdos.

É dada ênfase ao proporcionar uma forma de avaliar os utilizadores, não só através das suas acções, mas também pelo conteúdo criado, tendo como hipótese que tal será um factor de aumento do compromisso e confiança do utilizador. É feita uma exposição dos métodos de cálculo para a tal avaliação e diversas simulações para estes métodos.

A plataforma web resultante define uma base para exploração de diversas abordagens na criação de conteúdo e de partilha colaborativa. Os casos de uso do sistema são analisados e discutidos, considerando este sistema como alicerce para aplicações web, focadas na colaboração e estudo.

Palavras-chave: Aplicação web, Sistemas de gestão de aprendizagem, Aprendizagem activa, Colaboração, Partilha de conteúdos, Controlo de acessos, Avaliação

Contents

List of Figures

List of Tables

Listings

Glossary

Back-end Web back-end, or backend, refers to the server side of websites or web applications.

Fanout Fanout, or fan-out, is the process which delivers or spreads a message to one or multiple destinations. In notification feeds it is the process which pushes a notification / bit of data to all of the related followers, possibly in many small and asynchronous tasks.

Front-end Web front-end, or frontend, refers to the client side, to what happens on the web browser and what the user sees.

Hashing Hashing is the application of a one-way function to a variable sized input to produce a constant sized output. This function should be considered practically impossible to invert, not being able to retrieve the original input. This is used for safe password storage.

Responsive design Responsive design is a design paradigm for graphical user interfaces that adapt to different screen sizes. It should reorganize or restructure the user interface instead of scaling its size, making it adapt to smaller screens like it is the case with smartphones.

Web Used as an abbreviated designation for the World Wide Web, a hypertext system that operates over the Internet.

Acronyms

ACL Access Control List.

AJAX Asynchronous Javascript and XML.

API Application Programmable Interface.

ARIA Accessible Rich Internet Applications.

ARPANET Advanced Research Projects Agency Network.

CERN European Organization for Nuclear Research.

CMS Content Management System.

CRUD create, read, update and delete.

CSS Cascading Style Sheets.

DDP Distributed Data Protocol.

DOM Document Object Model.

HTML Hypertext Markup Language.

HTTP Hypertext Transfer Protocol.

HTTPS Hypertext Transfer Protocol Secure.

i18n internationalization.

IDE Integrated Development Environment.

IP Internet Protocol.

JSON JavaScript Object Notation.

JSP JavaServer Pages.

JVM Java Virtal Machine.

L10n localization.

LMS Learning Management System.

MathML Mathematical Markup Language.

MOOC Massive Online Open Courses.

MVC Model View Controller.

ODM Object-Document Mapping.

ORM Object-Relational Mapping.

RBAC Role-Based Access Control.

RDFa Resource Description Framework in Attributes.

REST REpresentational State Transfer.

SaaS Software as a Service.

SBT Scala Build Tool.

SOA Service Oriented Architecture.

SOAP Simple Object Access Protocol.

SQL Structured Query Language.

SSL Secure Sockets Layer.

SVG Scalable Vector Graphics.

TCP Transmission Control Protocol.

TLS Transport Layer Security.

UNESCO United Nations Educational, Scientific and Cultural Organization.

URI Uniform Resource Identifier.

URL Uniform Resource Locator.

VCS Version control system.

W3C World Wide Web Consortium.

WSDL Web Services Description Language.

WWW World Wide Web.

WYSIWYG What You See is What You Get.

XHTML eXtensible Hypertext Markup Language.

XML eXtensible Markup Language.

XUL XML User interface Language.

Chapter 1

Introduction

1.1 The Problem

In an educational context, managing content and tasks that are accomplished on multiple online tools has become a cumbersome effort due to the variety and number of Web applications. This happens with students, that usually organize digitally their study and work materials, and end up using several segmented cloud-based or device-based applications. Files, interactions with colleagues, ideas, scheduling and other bits of information are scattered throughout multiple applications and it becomes challenging to keep all that information organized in one place. Hence, there is a need for platforms that bring together the results of these highly segmented work efforts.

1.2 Motivation

Education is one of the main cornerstones for the civilization's technological and scientific advancement, being itself the aim of much research and development. Most education is provided by public or private institutions on a physical environment involving a teacher/professor lecturing several students, paradigm which is changing to a mix of face-to-face learning interactions and on-demand online learning [1].

The existing platforms for management of content are usually too closed or learners have limited access, in ways that restrict the ability of users to create content, communicate and collaborate. Sometimes these platforms can even work against their own purpose of providing learning content, yet there must be some ways to make content private between groups or individuals. The intended platform of this project is open in the sense that it can be accessed from anywhere in the world and everyone can sign up, which differs from some platforms that are deployed by an education institution and only their

students and staff can access it. Nevertheless, the availability of the platform depends on a hosting service. As such, although technologically open, its access may be constrained by business reasons. Being open does not imply that every content and group can be seen by not signed up users or guests. Owners of the content and groups can limit visibility and even hide groups and content from search results. It is considered important that users have their own private workspace where they can have personal notes and drafts, favourite content or organize and closely relate what is currently being used in a project.

Safeguarding privacy and management concerns of the institutions using the platform can be done in the context of Software as a Service (SaaS), even if it is most commonly achieved deploying some sort of Learning Management System (LMS), like Moodle, on private servers. Open access promotes public and communal learning, public scrutiny and improvement through better peer review, similarly to what happens on the open source software movement [2].

As these systems are usually deployed by the institutions on their own servers and closed to their students and staff, the adoption of these systems is dependent on the institutions, making them most suited for their needs of management, and sometimes less focused on the needs of students. The current generation of learners, even in continued education, is mostly familiar with computer and web related technologies and makes use of several other computer and web applications besides the ones provided by their educational institution.

Beyond the concerns of higher education, continued learning is also important. This type of education has been a concern for companies, so that their employees keep up-to-date with technology and technical knowledge. Traditionally continuous education is provided by companies. Yet, this is increasingly becoming up to the employees to educate themselves and keep up with new technologies, as companies provide less training.

1.3 Goals

Instead of providing a set of features directly related to education, this project focuses on the development of an infrastructure supporting: tags, access control, actions, rating and groups systems, which are hypothesized as important parts for making better educational software. The platform is developed with those goals set, possible approaches are analysed, and implications of these systems are reviewed.

1.3.1 Development Goal

This project focuses on providing a set of tools that bridge existing solutions for knowledge, social and content management, with special emphasis on education. It is a platform thought out for the needs of self-governed students, unlike most tools developed for online education (mainly LMSs), which focus on providing tools for the needs of the educational institutions. Although some of these systems provide good communication

channels between students and teachers, student's collaboration is one area in which few, if any, have achieved good results.

The platform should be provided as a web application, focusing on user groups, where most of the user interaction happens, and content gets created and shared. Groups are a virtual abstraction of a physical space for interaction between learners, or a community: with its members, settings, content and goals. Each user subscribes to or creates groups and has its own private workspace group. Making all these group "islands" work together is accomplished by creating group structures (organizing closely-related groups), and resorting to centralized communications tool, a topics or tags system, easy sharing, and navigation functionalities. This allows for multiple usage scenarios, increasing its potential usefulness for institutions and individuals, education and enterprise alike.

The underlying systems on the platform dealing with permissions, tags and rating make up a foundation for creating highly customizable groups and for developing other interesting features on top of it. One interesting feature, which these base systems make easier, is to link groups to content traditional online learning products and widely used tools like Dropbox and Google Drive. This would allow users to manage all their information in an organized and productive way. It also employs some engagement strategies, rating users and content through the feedback and actions of users, creating a sense of accomplishment in users and encouraging them to do more and better.

1.3.2 Research Goal

On a first stage, several different approaches to rating content and users are considered, taking into account several possibilities and considering which would be more effective on measuring the quality of content and the merit from user participation, on content and user rating, respectively.

The usability and features of the platform should be evaluated, using these questions:

- What use cases are met and how?
- Are features and systems understandable by users?
- Is the rating system fair and helpful?
- And other similar questions for each facet of the platform.

Ideally, data should be collected on the activities of users, yet the platform was not deployed for testing. Collecting data and surveying users would be important in order to seek evidence for the validity of this system. Ratings and feedback of users should be analysed in comparison with their perceived quality of the content, while trying to improve the ratings system upon this data. Another interesting measurements are user engagement and correlation between curating information and participation on creation content, with the user learning through that process.

1.4 Dissertation Structure

This dissertation highlights the efforts in creating each component of the web application. The present chapter gives some insight into the problem and proposed solution, giving perspective to some issues on the education technologies subject. This subject is analysed more in-depth in chapter 2, the state of the art. The state of the art features several topics related to the project, such as web technologies and collaborative environments.

Chapter 3 explains the core concepts of the project and its systems. Chapter 4 has further details on the computation of rating for users and content on the platform, as well as simulation of multiple algorithms for this computation. The technology stack, tools, overview of project structure and development approach is described in chapter 5. Further details of the system structure and implementation of each of its components are presented in chapter 6. Chapter 7 has a presentation of the implemented web application, while chapter 8 provides the conclusion of this document, discussing the project results and future work.

Chapter 2

State of The Art

In the following sections of this chapter, several approaches on relevant topics for the project, as well as some other related subjects, are studied and discussed. This should give a clear picture of current perspectives on web applications, their development, and employment of these on the field of education.

First, the state of the art of web applications is described, addressing how the World Wide Web has evolved technologically and in its applications. Then the solutions, tools and programming languages for the development of web applications is explored. After these technical subjects, a brief review is done on the usage of technology for educational purposes.

2.1 Web Applications

After the demonstration of the Advanced Research Projects Agency Network (ARPANET) in 1972, the interest in the research and technology communities has grown and the concept of Internet started becoming popular on many universities in the United States, mainly after the publication and implementation of both the Transmission Control Protocol (TCP) and Internet Protocol (IP) [3]. This early Internet stage did not have much public or commercial interest, at least until a widespread infrastructure was created, and the World Wide Web (WWW) became a much sought after application, as well as other applications like emailing [4].

Since the invention of the World Wide Web in 1989 by Tim Berners-Lee, a British scientist at European Organization for Nuclear Research (CERN) [5], there has been an effort to create websites with increasingly rich content to share information in an open and free way. That is why the WWW software has become public domain, in 30 April 1993. The core language of the WWW is the Hypertext Markup Language (HTML) and

its specification is currently on the fifth major revision [6].

The WWW technologies were initially created to provide access to static webpages. Their content could be any digital media, yet those contents of the HTML pages would not change much in these early web pages. Meanwhile, websites have become increasingly dynamic, both with dynamic behaviour on user interfaces enabling user interaction, as well as making it possible for users to create content. These efforts let users interact with each other, access real-time data, write and read comments, and so on. This harnessing of collective intelligence has become a basis for the concept of Web 2.0, a broad concept with a variety of different meanings, that include an emphasis on user generated content, content sharing, collaborative effort, and the usage of the web as a platform for generating, re-purposing and consuming data [7].

The innovative companies and insititutions that embarked on the Web 2.0 concept have been making their mark on the web [8]. Here are some examples of those that have embraced the power of the web to harness collective intelligence:

- **Google**, provider of the world most famous search engine, the giant of advertisement worldwide, dynamically uses page links, including on social software, to rank relevance of web pages;
- **Wikipedia**, an online encyclopaedia where any user can add or edit entries;
- **Flickr** and other similar sites, pioneering the concept of *folksonomy*, collaboratively categorizing sites with topics or keywords often referred to as tags (a move away from formal taxonomy);
- **Amazon**, huge retail business using user engagement as a main feature to increase sales;
- **GitHub**, platform for sharing source code repositories as open source projects or as private repositories; and many more of other popular websites.

In an effort to create complex application interactions and to support dynamic behaviours on web pages, a lightweight scripting language had to be chosen by web browsers to provide these functionalities. The language that became the web standard and available on all major web browsers is JavaScript. According to a survey from 2016 by the software developers community site Stack Overflow, JavaScript is the most popular programming language between web developers. It is used by 54.5% of back-end developers, 90.5% of front-end developers, 85.3% of full-stack developers and 55.4% by all software developers. [9].

2.1.1 Web Page or Application

Although very subjective, a distinction between websites, web pages and web applications should be made, defining those in broad terms, which will help further reading this

document. A **web page** is the client-side or front-end page shown to the user (runs on the client's web browser). Either websites and web applications use these to convey information to the user. The term **website** is traditionally related with a collection of web pages, accessible under a domain name (e.g., "www.google.com" or "www.wikipedia.com"). **Web applications** are applications containing rules, logic, data and interaction; they are not limited to just showing content and are more concerned with the back-end. The information of a web application is conveyed to the user through web pages and is located on a website.

2.1.2 The state of internet standards

The Internet has its foundation on several open standards, most notably the TCP and the IP. These standards are organized as a four layers protocol suite, the TCP/IP computer networking model. These layers are: link, internet, transport and application; from the lowest level (physical) to the highest level (more abstracted from the underlying network). The web pages or applications implemented on the WWW are only concerned with the highest layer, the application layer [10].

Communication on the WWW is done using the HTTP protocol, which is currently on its second major version. This protocol defines the requests and responses made on the internet: with an optional body with data and a header that holds the version of the protocol, the verb (a type of request), destination, response code and other data, for instance, the size of the request body. This protocol was designed as a modern software architecture for the web, to support Internet-scale distributed hypermedia systems [11].

On HTTP version 1.0, the verbs "GET" and "POST" were defined, which allowed for navigation on web pages, using "GET" to request pages and files and "POST" to submit forms and send other types of data. Version 1.1 added "PUT" and "DELETE", which together with "GET" and "POST" allow for the common data manipulation operations: create ("POST"), retrieve ("GET"), update ("PUT") and delete ("DELETE"). Note that this is not the complete list of verbs, just the major ones. The 2.0 version maintained these elements unaltered focusing more on compression an encryption. HTTP also allows the usage of non-standard verbs / methods [11].

As an open environment, the web community had to overcome privacy and security concerns. When sending HTTP messages over the internet, these can be read by any intermediary that may get access to sensitive information like passwords. This kind of security attacks are called *man in the middle* attacks [12].

To enable secure communications, the Hypertext Transfer Protocol Secure (HTTPS) was implemented. This sends HTTP messages encrypted over Secure Sockets Layer (SSL) or Transport Layer Security (TLS). The difference between the two is that TLS is the new and improved version of SSL. The encryption is done using one asymmetric key encryption (also known as public key encryption) algorithm. Asymmetric key encryption enables safe decryption to get the original message back. Both the server and the client

have a pair of keys, one public and one private [13]. These keys are mathematically related and the client keys are created through an handshake process with the server, which has an already defined pair of keys. The public key of the sender creates the encrypted message and the private key of the receiver decrypts the message back into the original. Public keys from servers should be signed and verifiable by a trusted entity. Consequently, HTTPS makes websites with sensitive information like user credentials much safer [14].

The WWW involves many standards not related with networking, but rather concerned with presentation and behaviour of websites. The implementation of back-end (server) systems is left open to implement the standards and protocols, for creating web servers with any language or tool. The combination of the front-end and back-end code compose the web application, which has been labelled as the full stack: the aggregation of technologies used to deploy the website.

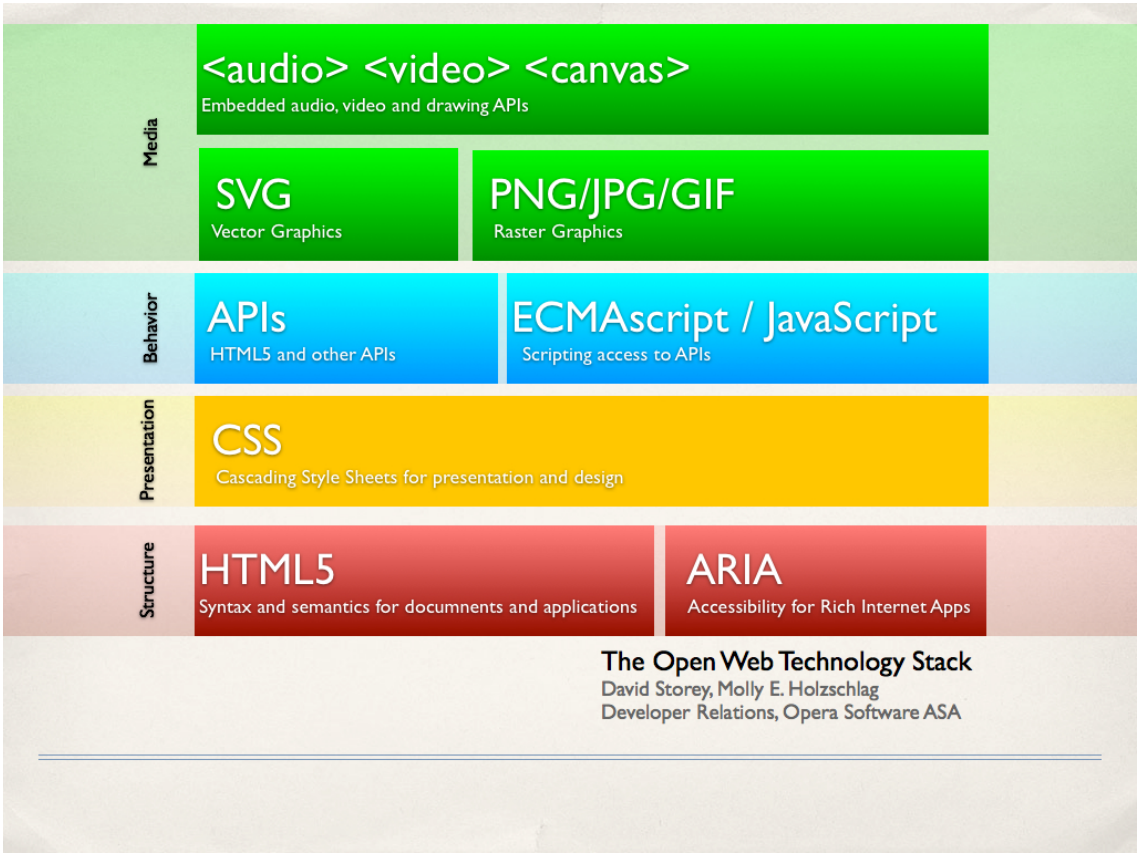


Figure 2.1: Map of web standards.

The standards illustrated in Figure 2.1 [15] have as the base layer the **structure** of web pages, which is defined by HTML. Several pieces of extra information can be used in conjunction with the HTML page structure: Accessible Rich Internet Applications (ARIA) defines context of buttons, which are the page navigation elements, description of images and other information enabling accessibility to people like the visually impaired.

The **presentation** layer on this stack concerns the usage of Cascading Style Sheets (CSS) to define the styling of the page, colors, position of elements, transitions, and so on. **Behaviour** refers to the programmatic features enabled by JavaScript, as well as some standard pre-defined libraries/Application Programmable Interface (API)s, also used by JavaScript. The **media** encompasses all media formats support on Web pages, as well as several embeddable elements for media that are bridged with corresponding APIs on the behaviour layer.

The World Wide Web Consortium (W3C) develops technical specifications and guidelines for these standards in an open effort to reach consensus, some results include: HTML, eXtensible Markup Language (XML), eXtensible Hypertext Markup Language (XHTML), CSS, Document Object Model (DOM), Resource Description Framework in Attributes (RDFa) and others. Other standards are not issued by the W3C, yet are compliant and/or aggregated into major standards: ECMAscript (standardized JavasScript), Scalable Vector Graphics (SVG), Mathematical Markup Language (MathML), glswoff, and many others. As stated on the W3C website, the "*W3C standards define an **Open Web Platform** for application development that has the unprecedented potential to enable developers to build rich interactive experiences, powered by vast data stores, that are available on any device. Although the boundaries of the platform continue to evolve, industry leaders speak nearly in unison about how HTML5 will be the cornerstone for this platform.*" [6]

Over time many developers have resorted to non-standard proprietary tools or languages to provide a richer experience to users using: Java applets, Flash, Silverlight or XML User interface Language (XUL), to name a few. One problem related with these various vendor-proprietary solutions is compatibility with web browsers and devices. This continues to be an issue as some browsers try to get ahead of the competition creating vendor-specific functionalities and proposing them to the standards later.

2.1.3 Model View Controller (MVC)

MVC is a software architectural pattern widely used in the context of web development, which separates responsibilities of logic, user interface, and data management. Some slight variations exist, like the model-view-presenter and hierarchical model-view-controller. Even inside the MVC pattern, some details like separation of model and controller change slightly [16].

Despite different interpretations of this pattern, the interaction between its components is represented in Figure 2.2 [1]. The MVC components are divided in the following manner:

- **Models** – manage the application data; each model represents a data aspect and contains logic concerning its data manipulations and rules.
- **Views** – these are used as the output of information; they include graphical user interfaces, but can be any output data.

[1] https://commons.wikimedia.org/wiki/File:MVC-Process.svg

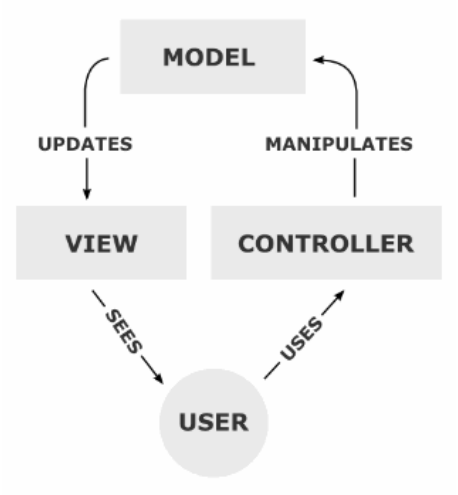


Figure 2.2: Interaction between MVC components.

- **Controllers** – handle input, manipulating and serving the necessary views and models.

2.1.4 Service oriented architecture

As defined by the W3C, a *"Web service is a software system designed to support interoperable machine-to-machine interaction over a network"* [17]. The protocol supported for implementation of web services is the Simple Object Access Protocol (SOAP). It relies on XML as the messages format and service definition file, Web Services Description Language (WSDL). Another approach, which is not a protocol, is the REpresentational State Transfer (REST). REST does not enforce message format or strict service definition; it is simply an architectural style, usually sending XML or JSON messages over HTTP or HTTPS. As Figure 2.3 shows, there is a rise of web services implemented with REST, yet SOAP remains much relevant on enterprise software as, opposed to REST, it enforces structure and security measures [18].

Web applications with the usage of some sort of Service Oriented Architecture (SOA) are becoming the norm in web development. Services can be used either internally, between multiple services running on the Back-end; or externally, with the access to the external third-party services being done by the Front-end or by the back-end code. A service is usually defined as a self-contained application. Exposing a service with which other applications on the web can communicate, through an API, is a web service. The defined set of entry points, their related actions, input and output is called the API. Web services are capable of: being utilized by other applications, possibly adding functionality; interchange data between services; implementation of the application for other types of clients, such as mobile device applications (also known as APPs)[19].

Following the rule of "separation of concerns", SOA can be the basis for the implementation of a web application, with the core logic of the application as a web service

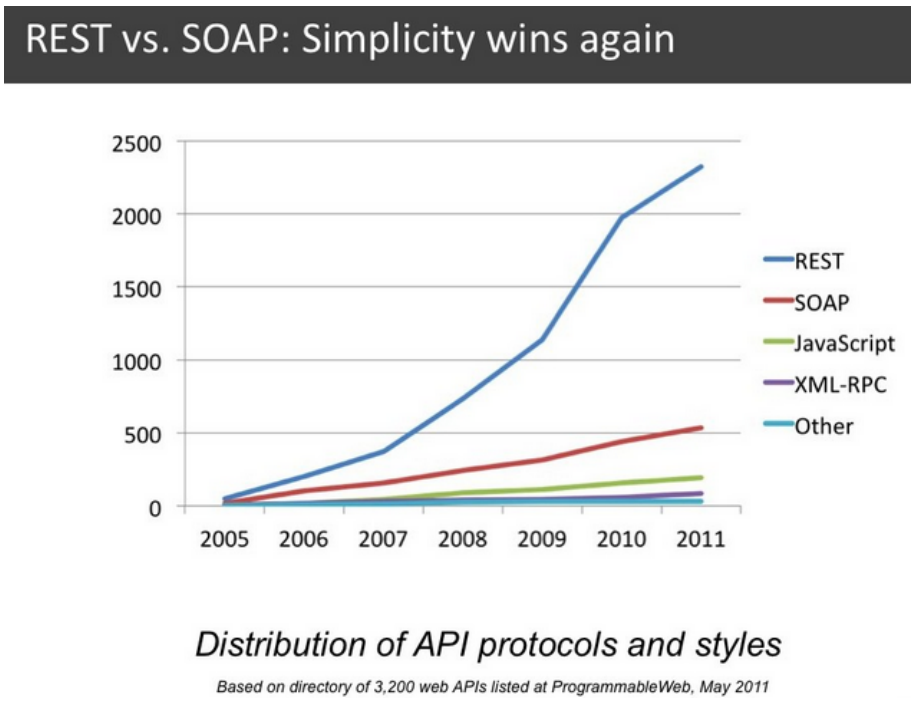


Figure 2.3: Web service APIs by protocol or style [18].

and the user interface as a decoupled application running on the client. This has become a trend in web development, usually providing an interface with the use of a JavaScript MVC framework on the front-end, which asynchronously fetches data from the server, typically as JSON documents.

The recommended best practice for building scalable web services is to develop small services (sometimes called micro-systems) instead of a monolithic architecture service. This allows for easier scaling and reuse of the components, yet it slightly increases overhead, mainly concerning serialization of data. It also adds some complexity, paying off with easier maintenance for larger applications [20] [21].

Implementing web services using HTTP for communication and Uniform Resource Identifier (URI) (commonly called URL) for their resources (data) is defined in the widely used REST architectural style or model. The HTTP verbs defined on the specification include the most important data manipulation operations, represented by the acronym known as create, read, update and delete (CRUD), which allows the development of clear service APIs for data exchange. These action support most of the use cases for web services. Another step for building a clear API for services are best practices for URIs: resource locations should be named and structured appropriately. Note that Uniform Resource Locator (URL)s are a subset of URIs, so both terms can be applied. The convention for the routing to resources [22] is detailed as follows:

- **(GET/POST) /users**: Gets a list from the *users* **collection** or creates a new record.
- **(GET/PUT/DELETE) /users/3**: Get, update or delete the **record** identified as *3*.

- **(GET/POST) /users/3/groups**: Get the collection of *groups* **associated** with *user 3*, or create a new record with the association.
- **(GET) /users/3/groups/1**: Get the *groups 1* record associated with *user 3*, usually the other operations are done through the canonical route for that record (**/groups/1**).
- **(POST) /users/3/promote**: Runs **custom action** "promote" to the *user 3* (for when CRUD actions do not suffice).

2.1.5 Real-time web applications

Applications that require high interactivity and prompt visual feedback require technological solutions that enable (almost) real-time communication between the client and server. This can be implemented in several different ways, all of them with their pros and cons. This is usually implemented for fetching data from a SOA application as JSON or XML, avoiding repeated retrieval of superfluous syntactic information about the web page, as this does not change, only its data does.

The most common and simple implementation of something similar to real-time behaviour is the use of Asynchronous Javascript and XML (AJAX) for reloading sections of pages or to fetch data. This is implemented through making requests (periodic and/or event-driven) from the client to the server. Other implementation approaches are the usage of: Comet, where the server keeps track of clients and sends data to the client; WebSocket, lightweight connection oriented protocol providing full-duplex (two way) communication streams on top of TCP; or other custom protocols like Distributed Data Protocol (DDP), real-time protocol used on the Meteor framework that synchronizes data between the client and server,etc. [23].

The mentioned synchronization of data is called data binding. Some front-end or full stack frameworks provide two way (full synchronization between clients and server) or one way (clients to server or server to clients) data binding. Some distributed or agent-based computing platforms on the back-end help in the development of these real-time behaviours, like in the case of Akka [2], integrated with the Play framework [3].

2.2 Web Development

With the ever-growing demand for more websites and web applications, the offer on development tools as become very wide and diversified. The set of tools for website creation encompass solutions from simple user-friendly graphical interface editors, with drag and drop for creating the page structure with desired components to create simple websites (like Adobe Dreamweaver), all the way to complex websites implemented using Content Management System (CMS) (like Drupal and WordPress). Regarding the development of

[2] http://akka.io/
[3] https://www.playframework.com/

web applications there are some efforts to make it more accessible, yet most of them are developed fully with source code on the desired language(s). Most of the help to create web applications (and some websites as well) comes from web development frameworks, which have a structure more or less defined for the project and a set of utilities to help in development like testing tools, security features, templates, caching, form validation, internationalization (i18n) and localization (L10n) [4], database management, and many other features.

These tools have been changing over time keeping with the industry and user demands, as well as design models that have continuously evolved, most notably the MVC and SOA models described previously. Some of these tools focus only on a specific aspect or a niche approach of web development, or solely on front-end and therefore more graphically oriented development, or on back-end development, while others try to provide a full-stack tool.

Not strictly a web development tool, the Integrated Development Environment (IDE), is an important part of the development process. Some solutions provide tight integration with the languages or frameworks, and even some helpers while designing web pages, like page preview and debugging features. The IDE used throughout this project is the IntelliJ IDEA[5] editor, which has integration with the Play Framework. Making use of the provided free license for educational use.

2.2.1 Software stack

Over time many bundles of the same software solutions for Web development have become widely adopted. These are often available as single installation packs, making it easier to setup the software that will serve as a platform for developers to create their applications on. These bundles are called software stacks or solution stacks, usually defined as combinations of operating system, web server, database and programming language(s). Some of the most common stacks are:

- LAMP, WAMP, MAMP and XAMPP, which are traditionally the most commonly used stacks, they stand for Linux, Windows, Mac OS and cross-platform as the operating system; Apache as the web server; MySQL or MariaDB databases; and Perl, PHP, or Python as programming or scripting languages. This stack is widely available as the standard commercial offer by shared servers, mostly for people that just want to create their static or simple website.

- MEAN – MongoDB database, Express.js as framework and web server, AngularJS as reactive front-end framework and Node.js as programming language. This stack

[4] Internationalization is mostly concerned with translation of text to other languages. Localization handles formats that are different, like in the case of date formats. The i18n and L10n abbreviations denote the 18 characters between the letters "i" and "n" in internationalization, and the 10 characters between between the letters "l" and "n" in localization, respectively.

[5] `http://www.jetbrains.com/idea/`

relies heavily on the JavaScript language, it is also cross-platform like some more modern stacks and is widely used in single page real-time web applications.

- And many other stacks, some even tailored for specific purposes, like OpenStack for cloud computing solutions.

Having well defined software stacks, sometimes easy to configure, facilitates the sharing among developers of: application modules, libraries, starter templates, open-source solutions and others. This also facilitates migrating projects between machines.

Software stacks may include web development frameworks, like in the case of MEAN, including the Express.js and AngularJS frameworks (for back-end and front-end, respectively). Most back-end web development frameworks include a web server, work with several different database solutions and possibly support multiple operating systems. This makes the frameworks an important part on the software stack.

2.2.2 Back-end programming languages

For server-side scripts, the most common language used is **PHP** [6], used by most websites, 81.9% according to W3Techs [7] [24]. PHP is an open source general-purpose server-side scripting language and has a huge online community. It is typically used in websites with smaller scale, yet some large-scale applications like Facebook use it in conjunction with other technologies to make it a more scalable solution. PHP is sometimes regarded as outdated by many web developers, yet updates to the language and many modern PHP frameworks have high performance while providing all common features of modern frameworks.

ASP.NET (C#) [8] has much usage in the Microsoft community, being the primary technology solution for Windows servers and on Microsoft Azure. From the most common server programming languages this is one of the few that are not open source and the least portable solution.

Java [9] is one of the main server-side languages for enterprise software. It is an high-level object oriented programming language that runs on top of a virtual machine, the Java Virtal Machine (JVM). It is highly portable as there are implementations of the JVM that run on most computers. Some websites like Amazon and the Apple App Store use Java-based web frameworks. One important aspect to note is that compared with lightweight scripting languages, the use of the JVM brings greater overhead and minimum system requirements to web servers. Also part of the Java technology family, JavaServer Pages (JSP)[10] enables web development directly on the Java language.

[6] http://php.net/
[7] http://w3techs.com/technologies/overview/programming_language/all
[8] http://www.asp.net/
[9] https://www.oracle.com/java/index.html
[10] http://www.oracle.com/technetwork/java/javaee/jsp/index.html

Scala [11] is a recent language that also runs on top of the JVM, yet it has both object-oriented and functional programming. It enables to mix Java and Scala classes, taking advantage of the Java extensive collection of libraries, frameworks and tools.

Node.js [12] – Given the rise of popularity of the JavaScript language, dominating as the front-end web development language, it started being used in other client applications. One example is Adobe AIR, a cross-platform desktop runtime created by Adobe that allows web developers to use web technologies to build and deploy Rich Internet Applications (RIAs) and web applications to the desktop [25]. Node.js, an open-source JavaScript runtime, initially developed by Google, has become one important solution for back-end development.

Python [13] and Ruby [14] are also popular server side programming languages, Ruby being mostly known for its popular web development framework Ruby on Rails. The Ruby language is the least familiar of these languages. Python is a scripting language with a very readable syntax and easy entry-level, providing several frameworks or libraries for web development, like Flask and Django.

2.2.3 Web development frameworks

The landscape of web development is extremely diverse. There is a great amount of languages, libraries and frameworks for back-end development. For the front-end there are less options on programming languages beside JavaScript, yet there are some languages that are converted into JavaScript and/or run in some browsers: CoffeeScript, TypeScript (Microsoft web browsers), Dart (Google browsers), to name a few. All templating languages are converted into HTML or XHTML. For page styles there are also some preprocessing languages like SASS and LESS that are compiled into CSS. These add complexity to projects and their workflow, yet on big projects languages like SASS provide tools like calculations and managing colors that CSS does not natively provide.

Full stack frameworks usually combine both the readily available functionalities of back-end frameworks and front-end interpreted languages or templating languages, mixing front-end languages with back-end elements or vice-versa. Frameworks are sets of libraries provided in one or more programming languages, becoming part of the software stack of a system.

Some complex web applications development has deviated from traditional PHP, HTML, CSS and JavaScript implementations. Using other languages or frameworks that are translated into HTML, CSS and/or JavaScript, may offer a greater level of abstraction. These higher level abstractions allow for overcoming hurdles like boilerplate code, multi-browser compatibility and lacking readily available functionalities. Note that boilerplate code is repetitive code that is necessary.

[11] http://www.scala-lang.org/what-is-scala.html
[12] https://nodejs.org/en/
[13] https://www.python.org/
[14] https://www.ruby-lang.org/

These frameworks already include many of the needed tools and functionality, such as authentication, internationalization, dynamic templates, database integration, routing, server deployment and configuration, security, asynchronous and/or non-blocking I/O, API generation, connection and streaming support, modular / pluggable and much more. Some of the contemplated frameworks for development, offering slightly different features and development approaches:

- Django [15] (Python) – Back-end framework.
- Play Framework [16] (Java / Scala) – Back-end framework, Akka.
- Express.js [17] (Node.js) – Back-end framework.
- Meteor [18] (Node.js) – Full stack, real-time using own protocol DDP.
- Nitrogen Web Framework [19] (Erlang) – Back-end framework.
- Fat-Free Framework [20] (PHP) – Light-weight back-end framework, improving upon PHP features.

2.2.4 Database solutions

In order to store, retrieve and edit data, there is the need to use a database solution on web development. Even if the data is stored directly in a file system, it still uses some kind of indexing or labelling of the data. Database management systems are solutions that handle these concerns.

One of the challenges encountered while designing this application was the diversity of object types, like content and metadata, that would be stored on the database and relate with each other in a very unstructured way. Traditional Structured Query Language (SQL) relational database systems are more adequate when data is more structured, both in terms of schema and relationships, which is not the case in the project to accommodate different types of content and groups. Currently there is a surge of new database solutions suited to new challenges, such as big datasets, massive scale and data analysis that do not use SQL, called NoSQL databases.

MySQL is a SQL relational database solution. It *"is the world's most popular open source database. With its proven performance, reliability and ease-of-use, MySQL has become the leading database choice for web-based applications"* [21]. It also offers several developer tools like MySQL Workbench [22], allowing to design data models to generate SQL code and vice-versa.

NoSQL is defined as the database systems that do not use the language or relational-oriented organization of SQL. Two interesting types of NoSQL solutions explored for this

[15] https://www.djangoproject.com/
[16] https://www.playframework.com/
[17] http://expressjs.com/
[18] https://www.meteor.com/
[19] http://nitrogenproject.com/
[20] http://fatfreeframework.com/
[21] https://www.mysql.com/about/
[22] https://dev.mysql.com/downloads/workbench/

project are **document oriented** and **graph oriented** databases.

CouchDB [23] and MongoDB [24] are examples of document oriented databases. These document based database systems use the JSON format to store data objects as documents [26]. The MongoDB solution has clear instructions and use cases presented in its documentation, providing easy set up and process for getting started.

Cassandra [25] is a good example of a cross-over between flexibility of data schema and traditional table-based relational database functionalities, giving great emphasis on data consistency and integrity, while providing a degree of flexibility to the tables data schema. Yet, most features of this solution are thought out and more appropriate for storing and analysing time-series data.

Finally, graph-based database solutions such as Neo4J [26], which are ideal for storage and representation of complex data structured as networks or graphs. This is useful for social networking data, allowing to query for friends of friends, and for applications like ontologies, with many intricate relationships (in this case synonyms and parent concepts). This solution was considered for storage and retrieval of the data related to relationships between users, groups and contents, yet these relationships were not the focus of development.

2.3 Learning, Education and Technology

A brief overview of the learning process is discussed in this section, separate from the more technology related section about online education and education technologies. This section has the purpose of supporting the approach taken on the platform design, as the efforts on the ratings, permissions and actions systems align with collaborative learning theories.

There are multiple notions of, or approaches to, learning and teaching. This is a vast subject for which no single methodology or approach has never been proven entirely correct. There are several scientific fields involved in the study of this subject. These are mainly related to epistemology, neuroscience, and psychology. Poor understanding of the emotional, chemical and neurological implications and requirements on the learning process, makes this an interesting and ever-evolving subject.

2.3.1 Learning theories

Here are some of the most important learning theories, listed by chronological order, summarized from an overview published by the United Nations Educational, Scientific and Cultural Organization (UNESCO) [27]:

[23] https://couchdb.apache.org/
[24] https://www.mongodb.org/
[25] https://cassandra.apache.org/
[26] http://neo4j.com/

- Behaviourism (early 1900s) – *"Learning is understood as the step-by-step or successive approximation of the intended partial behaviours through the use of reward and punishment"* [27];
- Cognitive psychology (late 1950s) – *"People are no longer viewed as collections of responses to external stimuli, as understood by behaviourists, but information processors"* [27];
- Constructivism (1970s and 1980s) – *"is the learner-centred approach whereby the teacher becomes a cognitive guide of learner's learning and not a knowledge transmitter"* [27], popularized by Jean Piaget;
- Constructionism (1980s) – *"greater focus on learning through making rather than overall cognitive potentials"* [28], by Seymour Papert, introduced usage of information technology and robotics into education;
- Social learning theory – *"suggests that people learn within a social context"* [27];
- Socio-constructivism (late 20th century) – *"knowledge is considered as situated and is a product of the activity, context and culture in which it is formed and utilized"* [27], by Lev Vygotsky;
- Experiential learning (early 2000s) – connected with constructionism, *"learning is about meaningful experiences — in everyday life"* [27];

Mostly after constructivism, learning theories seem to be adapting pedagogy to the information era we currently live in and to the usage of information technologies in education.

2.3.2 Learning Management System (LMS)

E-learning provided in universities all over the world traditionally uses LMSs such as Blackboard, Moodle and Sakai, which organize learning as courses. These systems are primarily designed for course management purposes and have limited impact on pedagogy. Furthermore, the life-cycle of interactions and content is limited in time, usually by semester. The limitations of LMSs, mostly lacking of collaboration features, do not support the new era of e-learning, with learning as a self-governed, problem-based and collaborative social process [29].

A LMS can be defined as a software system, usually web based, which enables the management and delivery of learning content to students [30].

The content management part of the system to be implemented should be a simplified hybrid between a CMS, a Document/Data Management System (DMS) and a Wiki system. Wiki systems, in essence, can be considered a CMS with a different set of features, own syntaxes and version-control centered architecture. These solutions are either simple,

with a very specific purpose, or very complex with lots of features trying to suit all needs. Some Enterprise Content Management tools combine these capabilities, yet they focus on business features and organizational processes (like Business Process Modelling), which are outside the scope of this project.

2.3.3 Massive Open Online Course (MOOC)

Currently the predominant model for delivery of education through the internet outside campus are the Massive Online Open Courses (MOOC)s. These are courses open for anyone. Top colleges at the United States of America started to provide their courses as a combination of short videos, quizzes and assignments, creating the format in which most MOOCs are currently provided. John Hennessy, president of Stanford University, provides some insights to the appearance, challenges and possible future of these courses:

> *"We started using technology to improve what was happening on campus. And we learned. We learned that if a video is simply a talking head with a set of PowerPoint slides, it can be just as stupefying as sitting in a big classroom listening to a lecturer. So what you need is more interactive learning. When Daphne Kohler started her experiment, she broke her lectures into chunks of 10 to 20 minutes and had a little miniquiz in between. And the miniquiz was able to identify that the student must have been asleep during that section or that there's one little thing to brush up on before moving on – hit this link and you'll go back to that place in the video. (...)*
>
> *We found that we were able to handle a lot of the Q&A through social networking. It's amazing that when you have 10 000 students in a class and a student puts up a question, the group quickly converges on the right answer: Several students put up an answer, other students come in and vote for what they think is the best answer, and there's a high probability that you'll converge to a pretty good answer in less than an hour."* [31]

As the scale of these online courses involves several tens of thousands of students enrolled, the assessment of their work cannot be entirely up to the course staff. This challenge for assessing projects of students in a large scale has been a problem with much research, mostly in the United States of America. Quizzes are easily implemented to be automatically evaluated, while some tools for automatic evaluation of projects with a well defined nature have also been created. The nature of these open courses also allows for crowdsourcing the evaluation process between the students. In conclusion, assessment in a large scale uses automatic and crowdsourced assessment, with a final evaluation done by the course staff [32]. This effort of automatic or collaborative assessment is in-line with the rating system purpose, taking a crowd-sourced approach to informal assessment.

2.3.4 Open education

Technology has been seen as both an enabler and a tool for learning. Many educational debates and much research have been made throughout history, either marketing new technologies as solutions to education's problems or disregarding some of these technologies. When paper was invented, it was disregarded by some as a medium appropriate for learning, as much of the information conveyed on the rhetoric and interaction of face-to-face lecturing would be lost. Yet, this would be one of the only ways, at the time, that all the world could become more knowledgeable, arguing that lecturing was too difficult to provide for the masses. The same revolution discourse happens as new disruptive technologies become widely available: audio, video, computers, internet, etc. [33]

Currently, the de facto approach to education gives great relevance to the use of computers, as these can convey information in written, oral and visual forms, therefore combining all the possibilities supported by previous technologies. Computer assisted education has developed greatly with the use of the internet, supporting mobile and distance learning. All these computer-based approaches to learning are called e-learning. On the extensive subject of e-learning, two of the most developed solution types are: LMS, used to create courses that are self-paced, for in campus courses support material or for online student evaluation; MOOC, mostly scheduled agenda courses that sometimes allow the student to achieve some kind of certification. These web-based large scale solutions for e-learning are attracting a lot of attention for university programs, continuing learning and employee training on companies [34].

Independently of the technology used by the learner, the most important aspect of the process is always the ability of the learner to acquire new knowledge and to accomplish project related tasks. Some of these tools for learning are related to active learning and collaborative learning, which are the main approaches delivered by the proposed project.

2.3.5 Collaboration

Collaboration is defined by Camarinha-Matos et al. [35] as *"a process in which entities share information, resources and responsibilities to jointly plan, implement, and evaluate a program of activities to achieve a common goal. It implies sharing risks, resources, responsibilities, and rewards, which if desired by the group can also give to an outside observer the image of a joint identity.*

Collaboration involves mutual engagement of participants to solve a problem together, which implies mutual trust and thus takes time, effort, and dedication. The individual contributions to the value creation are much more difficult to determine here".

Roy Soliman et al. [36] propose that there are eight essential ingredients for collaboration, which are: two or more people, shared space, time, common objective, focus on a goal, common language, knowledge about the goal and interaction. On their paper, these elements are explored in detail as a way of evaluating, in a technology-independent

view, collaborative environments and how these can be implemented to enable computer-enabled collaboration.

In collaborative networks the information flow is drastically different from that in public social networks [37]. As collaborative work is more focused on accomplishing tasks than simply broadcasting information as in public social networks, the information is forwarded through members relevant to the given task, while in social networks the information loses value as it spreads through the network, here the main value is in the task itself.

On a smaller scope, cooperation does not require to have joint goals, responsibilities and to be viewed as a joint entity. Cooperation usually is the highest level of interaction between users in most social software, unless there are circumstances of joint content creation, where collaboration might occur.

2.3.6 Social or collaborative learning

As already pointed out, social software has become a big part of the web with the Web 2.0 movement and in the same way it has been gaining adopters in education. Social software has a loose definition as software that facilitates interaction and collaboration among users. One example of a social software solution for e-learning, proposed by Du et al. [38], is based on a personalized user space containing their course network, social network and knowledge network, linking these in a way that enables users to build these during the learning process. It also features news feed, recommendations and search services, to facilitate navigation on the platform. It should also be pointed out that social software can stimulate learner-educator interaction and fosters a greater sense of community.

2.3.7 Acquiring and managing knowledge

Both LMS and MOOC solutions use processes for sharing knowledge with the students. Exploring knowledge creation and acquisition might lead to improvement of these features. The process of acquiring knowledge can be described as a five steps process [39]:

1. Knowledge **creation** – Information should be made accessible, easy to find and to use. The contents should be understandable and easy to create or edit, making it important to have a user-friendly content creator.

2. Knowledge **accumulation** – After the creation of content, collective feedback should add knowledge (i.e., comments, edit requests) and new relevant activities or information should be added or pointed out on the content. The authors of the content should be empowered to edit, control contributions and change the authors / contributors group.

3. Knowledge **sharing** – Communication between people should be facilitated. Giving a clear picture of interests, projects, groups and current tasks of users will help to find users to collaborate with and ease to pass on information and tasks. Suggestion of groups, users and content similar to the the user interests promotes interaction and creation of new teams too.

4. Knowledge **utilization** – The search for knowledge in the system, both as a quick search or an advanced multi-parameter search, should avoid noise and use the feedback collected for the content to highlight what might be best for the user. Another topic altogether is the creation of "remixes" of content that combines or changes the nature of the original material.

5. Knowledge **internalization** – This concerns the reiteration of knowledge and the ability of who acquires knowledge to improve upon it.

2.3.8 Informal and continued education

Informal or non-formal education is defined as education outside of a standard school setting. Informal education is predominantly applied to adult learners seeking to complement their formal education. Even as arguable its' value may be, there is a great undermining of the potential informal education has for completing and keeping knowledge current, which is becoming increasingly fast changing (explained with the rapid evolving skill-sets and job market). Adult learners usually work full-time, and they want to learn in their spare time or in an after work schedule. This makes an on-demand web platform suited for their needs, as happens with MOOCs.

Continued education lacks the research and commercial interest given to formal education for the young, particularly if outside the context of specific technical certifications. Yet, several MOOCs are capitalizing on the necessity of continued education for the fast evolving skill requirements of new jobs.

2.3.9 Mobile learning

Mobile learning uses mobile technology, like mobile phones or smartphones, to enable learning anytime and anywhere. Mobile learning is one of the current trends of educational applications, enabled by recent developments in wireless communications and mobile devices, allowing learners to engage in educational activities without being tied to a tightly-delimited physical location. In this review on the subject, by Wu et al. [40], it was observed that in 2010, higher education students accounted to 51.98% of mobile learning users, while adult learners[27] account for only 12.43% and disabled students for 0.56%.

[27] Adult learners work full-time or want to learn via mobile devices.

2.3.10 Education technology examples

Many technologies are currently used by students and educational institutions. These are too many to list here, yet they include:

- Discussion boards or forums, question and answers systems (Q & A), and threaded comments solutions. These are usually used in conjunction with a course page or MOOC for support and discussion of learning materials. Two examples of these are Piazza and Discourse.
- Learning management systems are offered with a varied set of features, some as open source software, while other are offered as proprietary paid solutions.
- Other applications are used for multiple purposes, like tasks organization (Trello), scheduling (Google Calendar), file sharing (Dropbox), and many more.

Chapter 3

Project Concepts

In this chapter several support concepts of the project are explained and their desired functionalities detailed. This gives an overview of requirements and design process before presenting their implementation.

3.1 Components Overview

The purpose of the planned system is to enable group-based collaboration in learning contexts, where users create and share content. Users are responsible for content curation, creation, and exploration, as well as management of these groups and content. As such, the main concepts of the platform are the **users**, **groups** and **contents**.

Additionally, several secondary components enable the features of these main ones, taking a support role. The overall architecture of the components or sub-systems of this project can be seen in Figure 3.1. Each of the elements from these main components (users, groups and contents) has associated the following objects:

- Metadata, using the **tags** system (section 3.2),
- **Access control**, based on user roles (section 3.3),
- Related **actions** and requests on the platform (section 3.4)
- One or more **notifications feeds** that show actions targeted to this element (as well as being followed by feeds) (section 3.5).
- Users and content also have a **rating** associated with them (section 3.6).

The navigation structure for the platform must enable search and exploration of these components and a well defined authorization system for users. The user authorization must discern between authenticated and unauthenticated users (logged in or public). Main features for user registration and authorization include: login, invitation, signup, email verification, email change, forgotten password, password change, user invitation

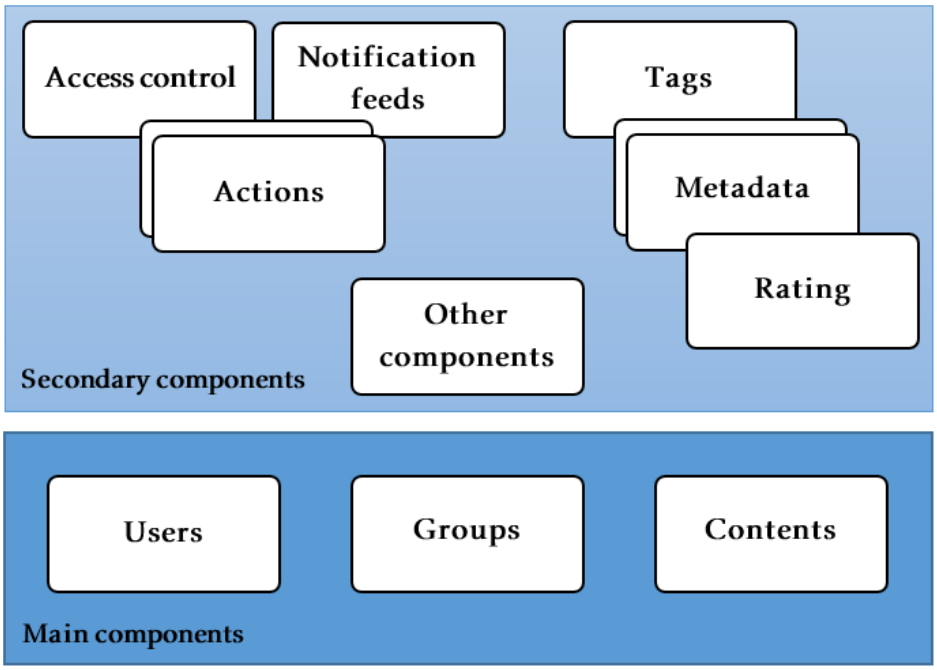


Figure 3.1: Diagram of the application logic components.

and user account settings. These features are detailed in Figure 3.2 and must be implemented considering safety, using cryptographically secure tokens for emails and user sessions.

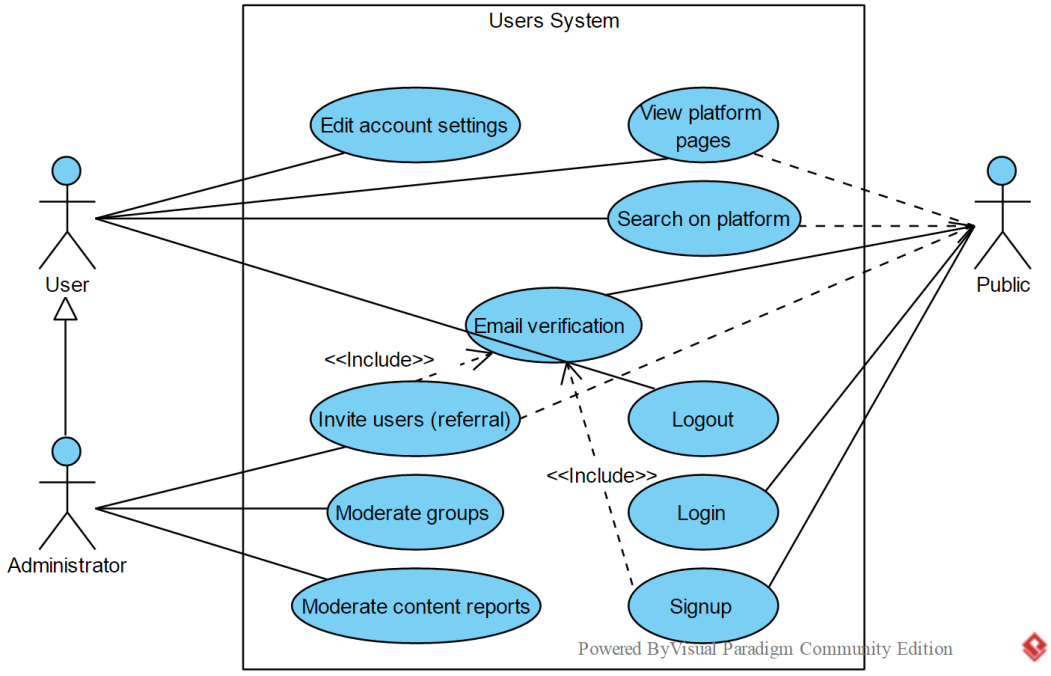


Figure 3.2: Basic and authentication features use case diagram.

First, a quick overview of these three main concepts (users, groups and contents) is presented, then the associated utilities are explained, and only after that the main components are detailed, already with an understanding of the other underlying features and systems.

3.1.1 Main concepts

- **Groups** are the main organizational structure on the platform. These hold content and place users together. They form a virtual space for user interaction.
- **Users** are the creators of groups, content and actions on the platform, so most sub-systems link to the users. Users also have the responsibility of managing groups and content.
- **Content** is the information created or being worked on and shared by users: notes, essays, pictures or any kind of files.

3.2 Tags and metadata

Tags define the types of metadata. These can be related to data components, like file extensions, topics, ratings, page views, any extra information that should be appended to an object. Each tag has a description detailing its purpose and behaviour, while metadata are the objects with the information attached to an object. In our case only main component objects have metadata. The tags sub-system should also allow users with administrative rights to manage custom tags, allowing the creation of tags like topics. The tags sub-system use cases are presented in figure 3.3.

This sub-system is an important feature for organization, search and even extensibility of the platform. For this, the tags sub-system should have some of the following features: structured in graph or tree, for enabling creation of ontologies; multi-typed, in order to have multiple different behaviours; multilingual and support synonyms, for easier usage. Tags are structured and identified using a namespace and predicate, being the namespace the category of the tag, not enabling by itself to create graph or tree structures between tags. This structure is called *triple tags*, in which metadata elements are comprised by three parts: a namespace, a predicate, and a value. This is exemplified in Listings 3.1 and 3.2. Values of metadata elements can be of any type, and further information related to the tag itself for relationships or languages can be added to the tag data model.

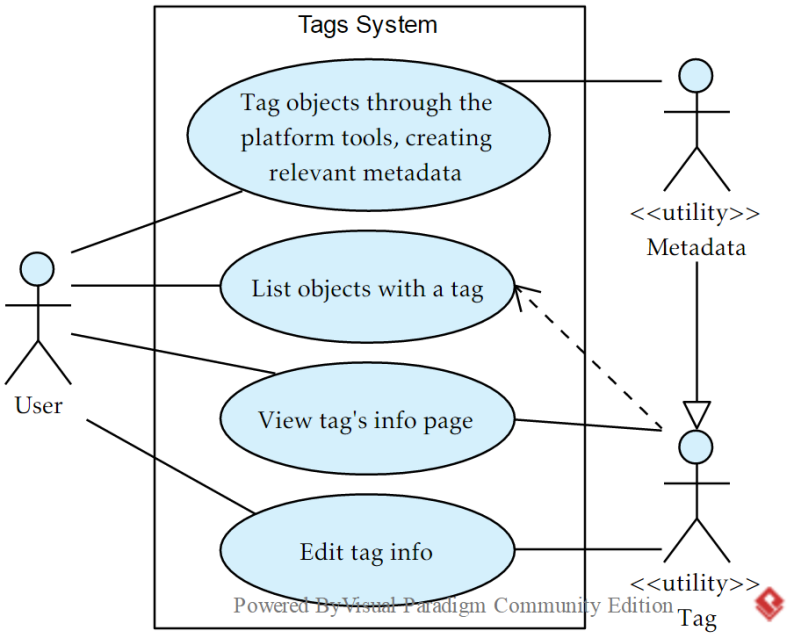


Figure 3.3: Use case diagram for tags sub-system.

Listing 3.1: Tag example

```
{ "_id" : ObjectId("56040a8474d5bc5d3dcbe864"),
  "tagId" : {
    "namespace" : "rating",
    "predicate" : "percentage"
  },
  "presentationName" : "Percentage rating value",
  "valueType" : "ANY"
}
```

Listing 3.2: Metadata examples

```
general:language="PT"
topics:example-topic={count: 2, value: [id1, id2]}
rating:percentage=56.3
```

The purpose of having a tags sub-system that can be structured as graph or tree is the creation of topics. These would be tags that define a specific scientific field, other subjects or even things like corresponding course modules (group as parent). The subject matter of groups and content would be defined through these topics, creating a *folksonomy* that would ease navigation and recommendation on the platform. This feature of relating and structuring tags is subject for future work, as it goes into complex subjects such as ontologies and managing those structures.

3.3 Access control and user roles

Each object on the platform main components has its own access control object attached, which defines the roles that the user needs to view (search and navigation), access (open) and do other operations with the relevant object. The user roles are divided into: global roles (distinguishing public from registered users), user roles (types of user), container roles (relates the user to an object), group member roles (types of group member). The access control defines the lists of user roles that have access to: visibility, access and other permissions to the object identified through a permission name. It also has a block list for specific users, to remove all permissions from that user.

The implemented case is a simplified access control, yet it could be the basis for more complex features, such as integration with authentication systems like LDAP and allow for delegated authentication. For example, the Faculty of Sciences and Technology from the New University of Lisbon uses the same LDAP based authentication system across their LMS solutions, Clip and Moodle. This delegation of authentication would, for example, allow to create a group that needs authentication in other platform. On such groups, users would be required to input their credentials on that platform, for those to be verified by an external authentication system.

Another simple and interesting access control system model considered for the project was the Access Control List (ACL). This model relies on listing each individual user that has each permission on objects, instead of listing the roles.

3.4 User actions and requests

The actions sub-system deals with representation and storage of user actions on the platform, as well as requests to do actions and their corresponding response. Possible actions were designed to accommodate two big requirements:

- make it possible to register user activity on groups and possibly use those data points in the computation of user rating;
- enable for users with no permission to apply an action, to still be able to request it; this request is received by users with proper permission to apply the action and let them decide if they want to accept or deny it.

This led to the creation of actions of multiple *types*: read, write, request and response. Actions also store in which group they were taken (*where*), *what* was the action (the action verb), *by whom*, *what for* (target object) and *about what* (data object).

Actions ended up being a big part of the access control system, when the user is trying to do an action the system checks if it is allowed to *read* or *write*, depending on the case. If it is not allowed and this is a *write* action, checks if it is possible to create a *request* instead.

The response mechanism for requests was designed to accommodate the possibility of using a voting system, to enable all users with different roles inside a group to vote, yet it would only need approval or rejection of a portion of the users to approve or reject the request. Users with full administrative rights could respond directly, but in the absence of one, or while it wouldn't respond this would allow for the other users to decide upon the request. This has many design implications on groups and adds much complexity, yet would be interesting for future work.

3.5 Notification feeds

Having the actions, with their requests and responses, happening in different groups on the platform, there is a clear need for a way to navigate through those actions. It is also important to be promptly notified of those actions or requests that would be considered more urgent. Common in social software, notification feeds are a main part of the collaboration process, keeping track of relevant events on the platform, aggregating similar events and usually making recommendations to the user in the process. In this case, notifications are more straightforward and functional: notification feeds follow a list of objects and in each of these *followings* there is a filter for which actions should go to the notification feed (the rest is ignored). So, a **following** is defined by a reference to an object (user, group or content) and the filter listing the relevant action types/verbs. Each action with the appropriate type targeted to an object with a feed **following** it creates, or adds to, a notification on that feed.

The user always has a *main notification feed* on the home page of the application, showing all actions targeted to the groups he is a member of, to contents he created and to himself. Other notification feeds are the *sidebar notifications*, meant for important notifications for the user, and *group inbox* showing requests made inside the group. The current implementation lacks some graphical interface elements to manage *followings* of notification feeds and to adjust filtered actions, yet it has the capability of replying to requests on the notification feeds. The notification feeds also have fields for defining if unread notifications should be sent by email and with which cadence (hourly, daily or weekly).

Feeds have no aggregation features, which are meant to simplify notifications. For example, aggregating notifications allows for just having one notification for multiple comments instead of one notification for each comment. The only case where information is added to notifications is to those concerning requests, adding the corresponding response.

3.6 Rating and feedback

User and content objects on the platform have a rating associated with them, showing the relevance and quality of each object, for the purpose of differentiating users and

contents by quality. The rating is generated using a set of assessable metadata elements. For content these are multiple types of feedback elements with different weights that are applicable to each content element and are created by the users.

Content rating value is calculated with the feedback elements from all different groups; users rating is calculated based on their actions and rating of their content. Both the content and user ratings have one raw value which is the counter of feedback weights and another value in percentage, to show on the interface a visual cue with this percentage rating.

In chapter 4 there is a comparison of multiple methods for calculating the rating for content based on the feedback and aggregating these to rate users. As content rating comes directly from feedback given by the platform users, the goal is to select which feedback elements should be evaluated on rating and how they are valued/weighted, as well as considering other factors like age of the content and recent activity.

As user actions are stored, they could be used to calculate the rating of users, yet these would make more sense if the user would have a rating in each group, making it possible to assert automatically if the user has been a proactive and collaborating user on the group. This is not done due to the complexity and ambiguity of some actions like editing and sharing content between groups, group size and different interest or relevance.

Further details about rating and its computation are described in chapter 4.

3.7 User

Users are defined as any registered user on the platform, not including public visitors, and these users need to authenticate with their credentials on the website. Unauthenticated public visitors of the website can still access the platform, yet they are restricted to see components that are set as publicly visible and/or accessible.

The user data model should allow for user management features like inviting, email verification and account deactivation. Other user information might be placed as metadata, which stores information like user rating. Safe user authentication is applied in conjunction with the user management and with the access control system.

Users also have a list of roles (as explained in section 3.3), a list of created content, and a list of groups with membership. The user rating should be shown as part of the user profile, as well as highlighting top user content, yet the later was not implemented.

3.7.1 Platform administrators

Some users can have administrative rights on the platform, being responsible for replying to reports on content, managing contents and groups. The platform has an optional mode where creation of new groups needs to be confirmed or rejected by one platform administrator.

3.8 Group

Groups were designed as multiple different types of pages and objects: projects, rooms, courses and subjects. Each of the group types would have slightly different interface elements or features. Groups ended up just being a generic container of users, content and tools like group chat. The concept of groups where tools can be added has the purpose to create a virtual collaboration space, similar to that of a class room. Ideally groups would have available tools, such as for connecting via audio or video between group colleagues, yet implementation or integration of those tools is a subject for future work. When creating groups, users can select from two editable access control pre-sets: public or private. Groups also have different types, each with its pre-defined settings, yet only generic and workspace groups are implemented. These types of groups are:

- **generic** – the basic empty group;
- **workspace** – usually private groups for users to bring there content from other groups, to be able to work separately. To work on their drafts or private content, each user always has a private user workspace of its own;
- **project** – an organized or individual effort towards a set of goals;
- **institution** – groups intended to have a set of courses as sub-groups;
- **course** – groups used to hold content related to a subject in a curriculum or any other extra-curricular course, holding information about the functioning of these, its materials, as well as the proper topics associated with it to easily find useful and related content.

A group is a container of tools and data, as well as a number of users associated to it through group membership. Group members can have several roles specific to the group, defining their permissions on that context. Groups can also have multiple tools or applications associated with them, as is the case with the group chat, further explained in the section 3.10, about messaging.

3.8.1 Content sections

Content sections are lists of content that are inside a group. The group members or visitors can navigate through this content, access, search and manage it, if they have the proper permissions. Content is displayed in a dynamic grid, each showing the thumbnail image, name, extension and rating.

3.8.2 Sharing content with groups

Instead of creating a special content section for content being shared to the group by outside users, usually labelled as an inbox, this shared content goes as a notification in

the group feed. Enabling users with administrative rights to approve or reject the content into the group, in a similar way as it would happen on a separate content section.

3.9 Content

The content sub-system is intended to encompass many forms of information: documents, code files, simple web URLs, editable pages, video streams, etc. Most of these can be placed as rich-text and links on pages or on files. Each of these content elements has several related metadata elements, containing its related data elements, like rating.

To improve the management of content, a content element has several kinds of user feedback related with it, which generate rating values. The different types of feedback elements are detailed in the chapter 4, dedicated to ratings.

Content elements have two different types: pages and files. Pages are rich-text HTML pages created by users, with the help of a graphical interface editor. Files are any kind of file that can be upload to the platform. Users with access to content can view it. In the case of files only image files can be viewed on the user interface, if the user has the required permission it can also download the content. As already mentioned, content is held on content sections, being the equivalent of folders in file systems.

3.10 Messaging

The platform has a simple **messaging** sub-system, which allows for communication between users and between group members. Therefore, a message always has the source user that has created it and a destination, either another user or the group chat (visible to users with permission to view the group chat, by default all group members). The messages have also a read or unread state to create visual cues for the user to know how many unread messages he/she has since that conversation was open. A **conversation** is the list of messages between two elements, so in the **user chat** the user has the list of all conversations and when that conversation is opened the messages are shown and marked as read. Messages are simple plain text, yet there could be attachments of all sorts, like files, platform content and even audio or video clips.

CHAPTER 4

Rating Content and Users

Rating is a broad term to define the quality of something[1]. It is also known as scores, feedbacks, referrals, or recommendations. A rating can be seen as an informal type of assessment and it is measured and presented in many different ways, e.g., 1 to 5 star ratings, 0 or 1 to 10 scoring, number of likes or votes, textual ratings and any sort of qualitative scale.

Ratings are an efficient and effective way of providing users with a measurement to assert the quality of content. Therefore an element to help building a trust relationship amongst peers in collaborative or open environments, as untrusted or even nefarious content will show up with bad ratings due to evaluation by users.

There are several concerns about uncertainties and biases on the rating value, which need to be mitigated as best as possible. Some users have a biased opinion or provide rating that is not honest, which brings some uncertainty. Another concern is that some high quality but niche content has a mediocre rating, just due to the small volume of its positive feedback. Liang et al. [41] hypothesise that ratings have two kinds of uncertainties: *"the uncertainty resulting from rating aggregation algorithms and the uncertainty resulting from other algorithm independent design factors, which are coined as* ***algorithm uncertainty*** *and* ***factor uncertainty****"*.

This chapter briefly studies some rating aggregation algorithms and assumes that feedback given by users on the platform is honest and matches the real content quality, so the *factor uncertainty* will be excluded from this analysis. The effort is directed towards selecting which algorithm and related parameters that best describe the given feedback elements.

[1] Cambridge dictionary definition for *rating*: a measurement of how good or popular someone or something is.

Note that rating algorithms differ from recommendation algorithms, as these do not evaluate the relation of the element to a context (another similar content, topic, search term, etc.). Therefore, this system could become more helpful for the user when combined with recommendation algorithms, highlighting contents, users or groups that are both similar and high quality. Yet, for the sake of simplicity no recommendation system is implemented.

4.1 Gamification and engagement

The use of reward mechanisms as done in games, but applied on other contexts is called *gamification*. In educational software, this is being implemented by many solutions, as a measure of incentive for the users. The main aspects of *gamification* are giving to users: points, badges, achievements or trophies, according to the user actions or performance, so that users measure their accomplishments, increase engagement and try harder. Another measurement of user's effort and performance are "leader boards" or similar features that allow comparison among users, highlighting or rewarding the best.

Especially in education, this need for engagement and motivation is true, yet it is controversial how much *gamification* should be applied. The most important goal here should be learning and engagement in the learning process, rather than "counting points" or collecting badges [42].

For these reasons, the user actions, their rating and rating of their content elements should provide important incentives for the users to measure their contribution in the context of this platform.

It should be left open the possibility of the platform to use other forms of *gamification* incentives, such as badges. The user's rating is also intended to ease the process of roles assignment, making it easier or obligatory for users to have a high rating to be able to have administrative roles, both at the platform and at the group level. These requirements to achieve better permissions could be also considered as incentives.

4.2 Rating examples

Here several methods for calculating ratings are presented. These methods are extremely diverse and usually companies do not disclose these details, as these might be a key component of their products or even open up the possibility for users to exploit the rating system to maximize their values.

Generic music playlist rating:

$$Rating = \left[\sum_{i=1}^{n} e^{-t}\right] * \left[max\left(0, log_{10}\left(\frac{u}{scale}\right)\right) + k\right] \tag{4.1}$$

This score equation is used as a formula to rate trending playlists on a playlist sharing website. It is based on snippet shared by the Lisbon startup company wave.cat [2], where there only are "up votes" on playlists. The first part of the equation, has the sum of all the **n** votes, with the value of exponential decay of time **t** of each vote. So votes weight for current time ($t = 0$) are 1 and as t increases this weight decreases exponentially. The second part of the equation places the count of unique users **u** that have been listening to the playlist (i.e., on the last month) on a logarithmic scale and use a scale factor for scaling according to the total number of users on the application (hundreds, thousands, millions, etc.). The "max" function is for the case where $u = 0$, avoiding negative values, and finally the "+k" gives additional points to recently created playlists.

This falls into the category of *popularity* ratings. Most ratings are based on the concept of what is popular recently. This is commonly used in rating news, music, discussions, etc.

Other rating examples:

Reddit news rating has a greater focus on recently added news, using the balance between positive and negative votes combined with time passed on a logarithmic scale. The calculation has several different branches, depending on the balance of votes [43].

Hacker news rating uses an algorithm based on the inverse function of the time (in hours), also affected by points/votes and a factor on time (called *gravity*) [44]:

$$\left(\frac{\#votes - 1}{(time + 2)^{gravity}}\right) \tag{4.2}$$

This *gravity* factor controls the speed at which the score decreases with the time passed, with default value of 1.8.

4.3 Goals for the Rating

In order to evaluate the results of the rating, a clear set of goals is needed. This will allow to determine which factors are taken into account, as well as their expected effect on the rating. Here is a list of the main goals set for the rating system:

- The rating, visible to the user, should be a simple bar – completely filled for very positive rating, half when neutral rating, empty with very negative rating.
- Make use of the feedback elements in a simple and effective way.
- The impact of the feedback elements should decay in time, or the age of content affect rating value.
- Give a boost to the rating of new content.

[2] http://wave.cat/

4.4 Rating Calculation

This section addresses the elements to create the ratings, types of feedback elements, how much each of these should weight and how the rating should be calculated.

4.4.1 Elements for evaluation

Content feedback types, i.e., direct content evaluation by users:

- Vote up or down – user liked or did not like the content (each user can only select one of these).
- Comment – user writes a text comment on the content.
- Edit request – user suggests edits for the content, for content owner to decide whether to update it or not (not implemented).
- Report / Flag – user reports content as inappropriate, nefarious or infringing copyright. Platform administrators handle these, if report is removed the rating penalty is removed, if confirmed the content is deleted.

Other elements that can be indirectly used for evaluating the rating value of content:

- Views – count of content page views by users and public.
- Age – days passed since content creation or last edit.
- Downloads – count of downloads of the content.
- Editions – content authors edit it.
- Shares – count of shares between groups inside the platform.

Other measurements for user engagement could be used on content pages. Several web analytics tools include measurement elements such as time spent on pages, events and clicks, but these are not implemented or used in this platform.

As the most important evaluation factor are the votes and these are just a binary value (yes or no). This does not allow for a more nuanced review of contents by the user. It would be interesting to have research data indicating the benefits and drawbacks of this approach.

4.4.2 Sum of feedback elements

One approach for calculating the ratings is to do a sum of the weights of each feedback, affected by its age. The effect of time is calculated for each of the feedback elements and summed, giving more relevance to recent feedback. This would only be beneficial in the case of content that was edited and therefore had its quality changed. Calculating a

weight for each element ends up being a resource intensive process, that needs to read all the feedback elements from data storage. First, the rating of content is calculated, then the rating for the users. This would be done through a scheduled event or a background working process, due to the need of updating.

In these equations (4.3 and 4.5), a measurement of **activity** (*activity.content* for content and *activity.user* for user) is used with the count of views of the page and actions related to the object, as well as a **scale** factor with the maximum **activity** value. In this user rating formula (4.5), not only the rating of the content created by the user is taken into account; other actions are also considered, in a similar way as the feedback elements affect the content rating.

The feedback and action weights (*feedback.weight* and *action.weight*) should be adjusted while testing the formula. This testing should be done by using a range of weights considered reasonable and evaluate the resulting rating values, comparing it with the expected value. The problem with this approach is that it is difficult to define an expected rating value, as it is something very subjective. This problem occurs in all of the rating calculations. It could be solved by comparing the perceived quality of content and its rating, interviewing users.

$$ContentRating = \left[\sum_{feedback=1}^{n} \frac{feedback.weight)}{log_{10}(feedback.time)} \right] * log_{10}\left(\frac{activity.content}{scale.content} + 1 \right) + k \quad (4.3)$$

$$action.value = \begin{cases} content.rating * action.weight & \text{for actions on own content} \\ action.weight & \text{other} \end{cases} \quad (4.4)$$

$$UserRating = \left[\sum_{action=1}^{n} \frac{action.value}{\log_{10} action.time} \right] * \log_{10}\left(\frac{activity.user}{scale.user} + 1 \right) \quad (4.5)$$

This approach becomes a problem when the content has many feedback elements. Those could be archived or pruned in order to facilitate calculation, but it is clear that traversing the feedback elements is a big drawback for scalability. Also as the rating value is dependent on the time, so the rating should be periodically calculated. Another notable drawback is the usage of the "**activity**" and "**scale**" factors, needing to hold counters for calculating these elements and making these calculations every time the rating is updated. Alternatively, the **scale** factor could be constant, but this does not accommodate cases where content is in groups of different dimensions, unless there is a way to scale for each group. The "k" would be a value for boosting rating of new content. To encourage the creation and curation of good quality content, with few or no errors, the down votes have more weight than the up votes.

Other rating measurements have been considered besides the *content* and *user* ratings, to provide information for users. Those other rating measurements were: *activity*, *group*,

content and *user* on the context of groups, and *discussion*. *Discussion* rating would be exclusive to users, showing how much other users comment on this user content and how much the user participates on comments of other content. These ratings were not implemented, as they either add superfluous noise information (activity and discussion are part of the feedback), or are complex (ratings relative to groups or group structures).

4.4.3 Counter based weighted average

The problems observed on the formulas presented in section 4.4.2 can be overcome by using a counter that holds the sum of the feedback element weights. This uses the feedback weights without considering when they were created. This enables for a faster calculation as the previous sum algorithm, using the already stored value which we call the *raw rating value*, just adding the feedback weights when the rating is re-calculated.

As the feedback elements of each type will have the same weight, this formula can be written as the sum of the weight times the count of each feedback type. Here is the weighted average formula: $\sum(\#feedbackElements.feedbackWeight)$

4.5 Simulation of Aggregation Methods

In order to verify that the goals for the rating system are met, a simulation was created to test multiple algorithms. This simulation generates content feedback randomly for 17 content elements. This amount of contents, 17, is sufficient for comparison between multiple cases of content quality. Also these content elements are assigned to users, in order to have some users with multiple content elements. The assignment of content to users is random, the content number 1 always belongs to user number 1, the next (number 2) is from user number 1 or 2 (50% chance each), and so forth. This way there are from 1 up to 17 users in each simulation.

The simulation also generates the age of each content and its views, enabling the comparison between algorithms that use this information and those that do not. This data is shown in Figure 4.1, as used on the first simulation run.

For the 17 content elements, 440 feedback elements are generated and randomly assigned to each content element. This gives an approximate proportion of 25 feedback elements for each content. It can be considered a low amount of feedback, yet this is proportionate with the low amount of contents and users in the simulation, and also allows for a clearer correlation between the feedback elements and the resulting rating. The type of feedback element is also random, but depends on different probabilities for each type. This assumes that there is more emphasis on positive feedback, having more up votes than down votes and few reports. The probability for each feedback type is as follows:

- Up vote - 30 %
- Down vote - 23 %

user id	content id	time (days)	views
1	1	99	271
2	2	96	366
2	3	32	457
3	4	56	278
3	5	46	432
3	6	104	85
4	7	84	67
4	8	23	1056
5	9	19	220
6	10	24	549
6	11	93	702
7	12	13	261
7	13	51	222
8	14	70	206
9	15	75	1069
9	16	90	264
9	17	16	1019

Figure 4.1: Users and their content elements, on simulation run 1.

- Favourite (number of groups) - 10 %
- Comment - 25 %
- Report / Flag - 0.5 %
- Download - 11.5 %

This allows to check if the rating value is what would be expected, given the existing feedback elements, always assuming that the feedback reflects directly the content quality. Several algorithms are simulated, all using the same users, contents and feedback data, excluding complex and computation intensive algorithms that are not cost-effective [45]. Also, due to complexity and lack of support on the web application, group-sensitive ratings were not created.

4.5.1 Aggregation algorithms

The explored aggregation algorithms include averages, weighted averages, beta probability density function, half weighted average, probability distribution based, weighted majority algorithm, or combinations of these. Some of the aggregation algorithms listed above use variables that were not considered on the design of the application, therefore are not used on the simulation. These excluded algorithms include parameters like the weight of each feedback (affinity between users) and feedback that directly rates the content qualitatively (as scoring from 1 to 10). Concerning these restrictions of the system and evaluation of the cost and effectiveness of these algorithms [45] [46], the selected aggregation algorithms were:

- Simple average: $\frac{\#votesUp - \#votesDown}{\#votesUp + \#votesDown}$

- Beta probability model: $\frac{positiveFeedback + 1}{positiveFeedback - negativeFeedback + 2}$ or for simple votes: $\frac{\#votesUp + 1}{\#votesUp + \#votesDown + 2}$
- Weighted average: $\sum(\#feedbackElements * feedbackWeight)$
- Weighted average with content age: $\frac{\sum(\#feedbackElements * feedbackWeight)}{log_{10}(1 + contentAge * 1.7)}$
- Weighted average with content age and content count factor: $\frac{\sum(\#feedbackElements * feedbackWeight).(log_{10}(\#contentElements * 0.7) + 0.3)}{log_{10}(1 + contentAge * 1.7)}$

Liang et al. (2005), provide useful insights into several rating aggregation algorithms, yet these calculations use discriminated weights between users with their trust or "relationship weight". This overview dismisses some computationally intensive algorithms, as their performance is not usually better than simple averages and probabilistic models [45].

From these methods of aggregation of the feedback elements into a rating value, several different approaches are used to create the percentage value to use as a visual cue on the platform interface. For the relative rating value in percentage, several approaches were used to transform the aggregation result into a percentage value, between 0% and 100%. Note that some aggregation calculations like the simple average and beta probability model already give out a percentage as result, yet are not to scale as the simple average gives negative values too. Here is a list of the formulas chosen to calculate the percentage value:

- Using average or beta function to calculate percentage (simple average needs transposing from -1 to 1 scale into 0 to 1 scale);
- Arctangent – the arctangent function turns a value from negative to positive infinity into a value between negative and positive $\frac{\pi}{2}$, which can be easily scaled and translated into the percentage range;
- Linear interpolation – simple linear interpolation value between the highest and the lowest raw rating values (overall);
- Average between arctangent and linear interpolation.

4.5.2 Feedback types and weights

On Table 4.1 the feedback types and each of their weights is listed, two different weight values are used. The **Weight** values were selected considering the relevance of each feedback type and after testing their effect on rating, through the simulation, they were slightly changed resulting in **WeightAlt**.

[1] User can only either vote up or down, never both.
[2] These are removed as soon as the requests are removed by an admin user.

Table 4.1: Feedback types.

Feedback type	Weight	WeightAlt	Count/User	Description
Vote Up	1	1	*1* [1]	User likes the content
Vote Down	-0.9	-1.1	*1* [1]	User does not like
Comment	0.1	0.05	*N*	Text comments
Share to Group	0.4	0.4	*N*	Groups shared with
Download	0.05	0.025	*N*	Counter of downloads
Views	$\log_{10}*0.2$	...*0.1	*N*	Counter of page views
Edit Request	-0.5	-1	*1* [2]	Request for editing
Flag	-3	-5	*1* [2]	Flag as inappropriate/spam

4.5.3 Parameters

The names of results for each of the methods are abbreviated in the simulations and its plots. These will be described and listed in this section.

Here the aggregation methods will be listed and respective names of these on the simulation:

- Simple average (**SimAvg**);
- Beta model of votes (**SimBeta**);
- Weighted average of feedback elements (**Aggr1Avg**);
- Beta model with weighted feedback elements (**Aggr1Beta**);
- Weighted average of feedback elements with effect of age of content (**Aggr2Avg**);
- Beta model for the second weighted average (**Aggr2Beta**);
- Similar to **Aggr2Avg**, but with the previously mentioned alternative weight values (**Aggr3Avg**);
- Beta model for the third weighted average (**Aggr3Beta**);
- Using the **Aggr3Avg**, but considering the number of content the user created, for the user rating (**Aggr4Avg**).

Names for the rating values adjusted for percentage value:

- Raw counter or percentage of positive versus negative feedback (**r or %**);
- Arc-tangent of the raw rating counter (**at**);
- Linear interpolation from user with lowest raw rating as zero and highest raw rating with one (**%i**);
- Average combination of the arc-tangent and the interpolation values ("at" and "%i"), as these provide very different measurements, of absolute and relative quality, respectively (**comb**).

This way, **rAggr2Avg** (**r** + **Arr2Avg**) is the raw rating value for the aggregation method of weighted average with content age.

4.5.4 Simulation run 1

On the first simulation run, the 17 content elements were assigned to 9 users. The aggregated ratings for each of these users will be analysed here, taking into account the attributes related with these data elements. The data relevant to this simulation run is presented in a summarized way in Table 4.2. This clearly shows several different scenarios:

- User number **3** has quantity over quality, creating three content elements (4, 5 and 6). Those have somewhat positive feedback, yet content 5 has a report, which should have a noticeable impact on the rating value, yet it should be a value slightly over 50%.
- User number **7** created two content elements (12 and 13) with good feedback and both are relatively recent, it should have a value well over 50% and the age sensitive aggregation formula should value this user more, relative to others.
- User number **8** created one content (14) element that has bad feedback, 13 down votes for 6 up votes and not many views. It is expected that this user will have a **bad rating**, below 50%.
- User number **9** created three content elements (15, 16 and 17), all with consistently positive feedback. The rating of this user should be the best on this simulation, as it has the best feedback on content.

user id	content id	votes sum	#votes	#groups	comments	reports	#feedback	#views	time (#days)
1	1	3	15	4	7	0	29	271	99
2	2	-1	11	2	9	0	28	366	96
2	3	3	15	2	9	0	30	457	32
3	4	0	14	2	5	0	23	278	56
3	5	4	10	2	10	1	28	432	46
3	6	2	10	1	9	0	21	85	104
4	7	0	16	2	8	0	29	67	84
4	8	-2	14	2	10	0	28	1056	23
5	9	5	13	2	4	0	23	220	19
6	10	0	10	2	11	0	26	549	24
6	11	4	8	2	5	0	17	702	93
7	12	3	15	3	6	0	28	261	13
7	13	4	10	2	3	0	18	222	51
8	14	-7	19	2	4	0	27	206	70
9	15	4	14	2	8	0	26	1069	75
9	16	5	13	3	15	0	34	264	90
9	17	7	15	3	3	0	25	1019	16

Table 4.2: Content and user information for simulation 1.

The following figures with the simulation result charts have the rating values on the horizontal axis and the user number on the vertical axis. For each of the users, a different value is shown for the rating. Each figure represents a different method to present the rating value: raw value (**r**), arc-tangent percentage value (**at**), linear interpolation percentage value (**%i**), and combined value with average between arc-tangent and interpolation (**comb**).

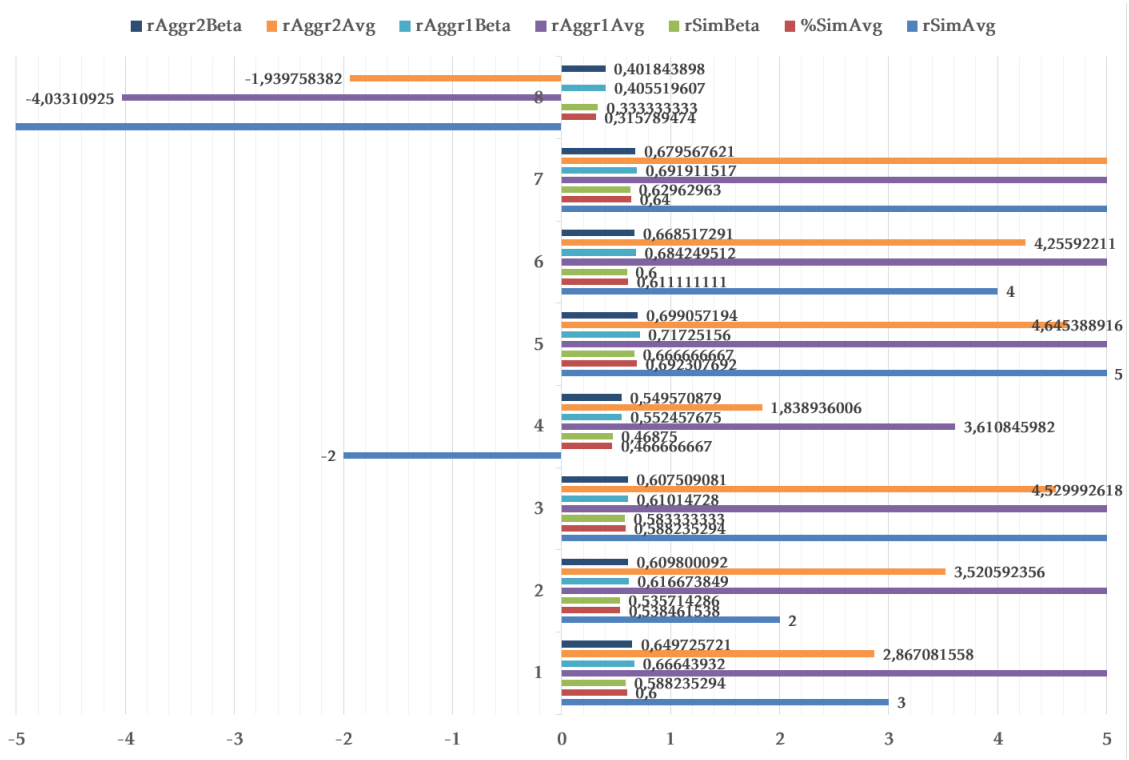


Figure 4.2: Raw values for user rating.

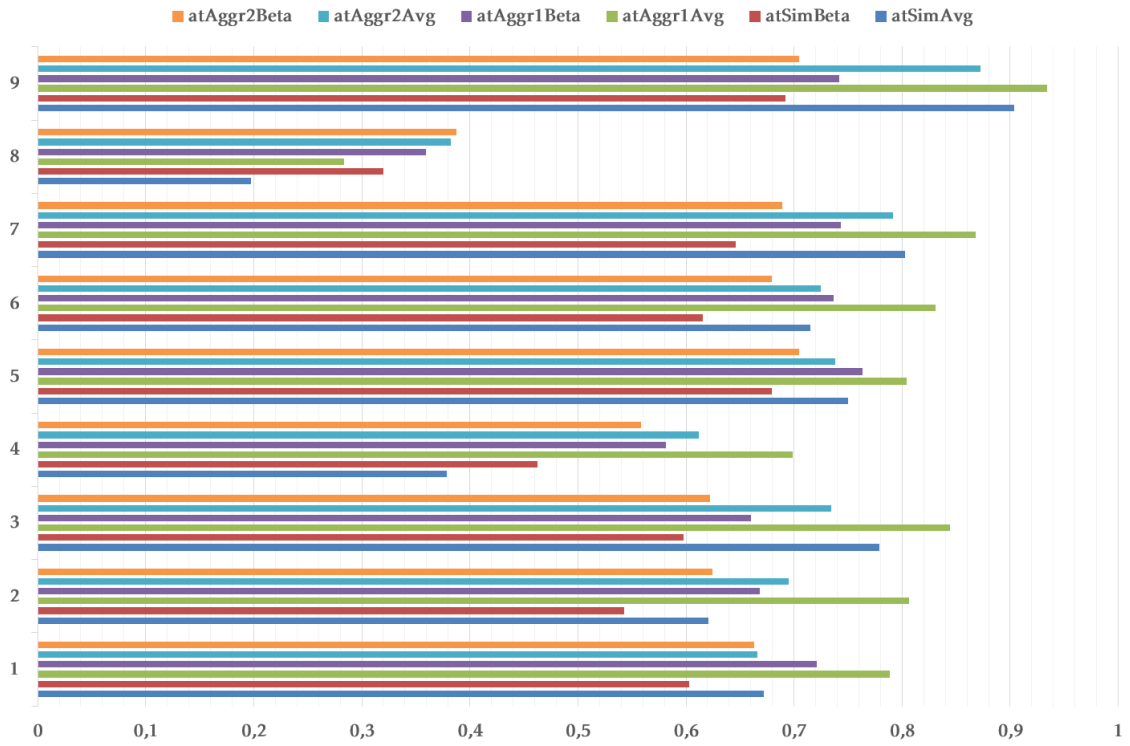


Figure 4.3: Arctangent values for user ratings (simulation 1).

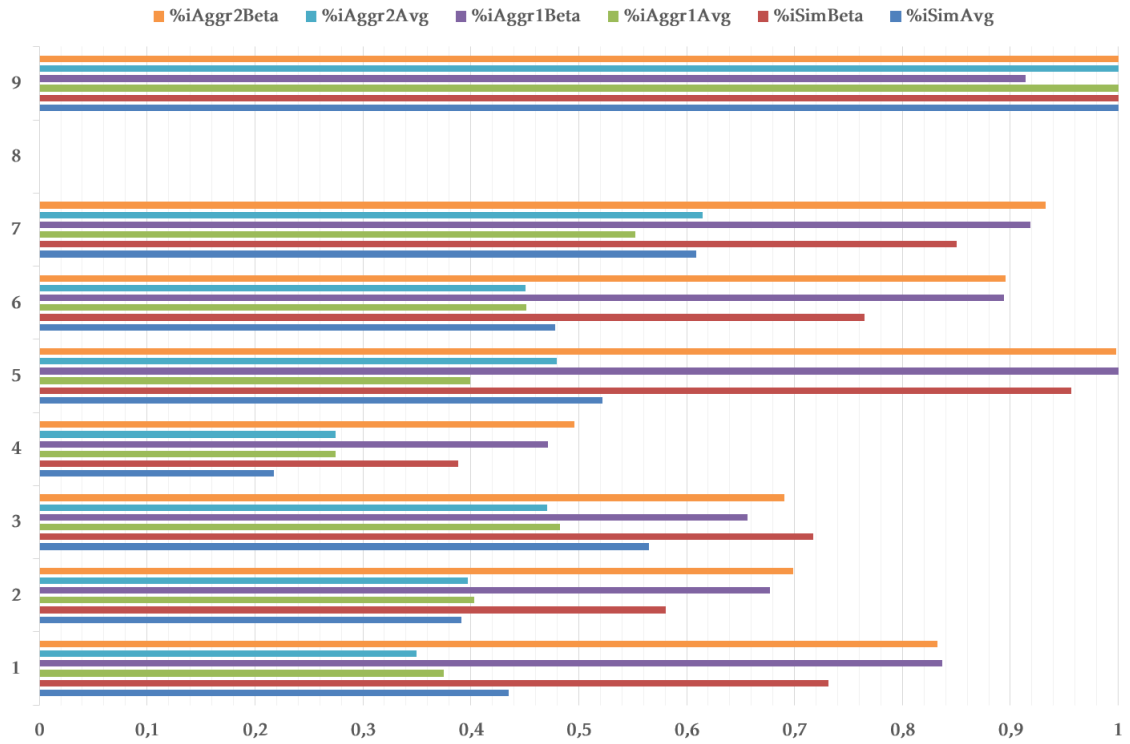


Figure 4.4: Linear interpolation values for user rating (simulation 1).

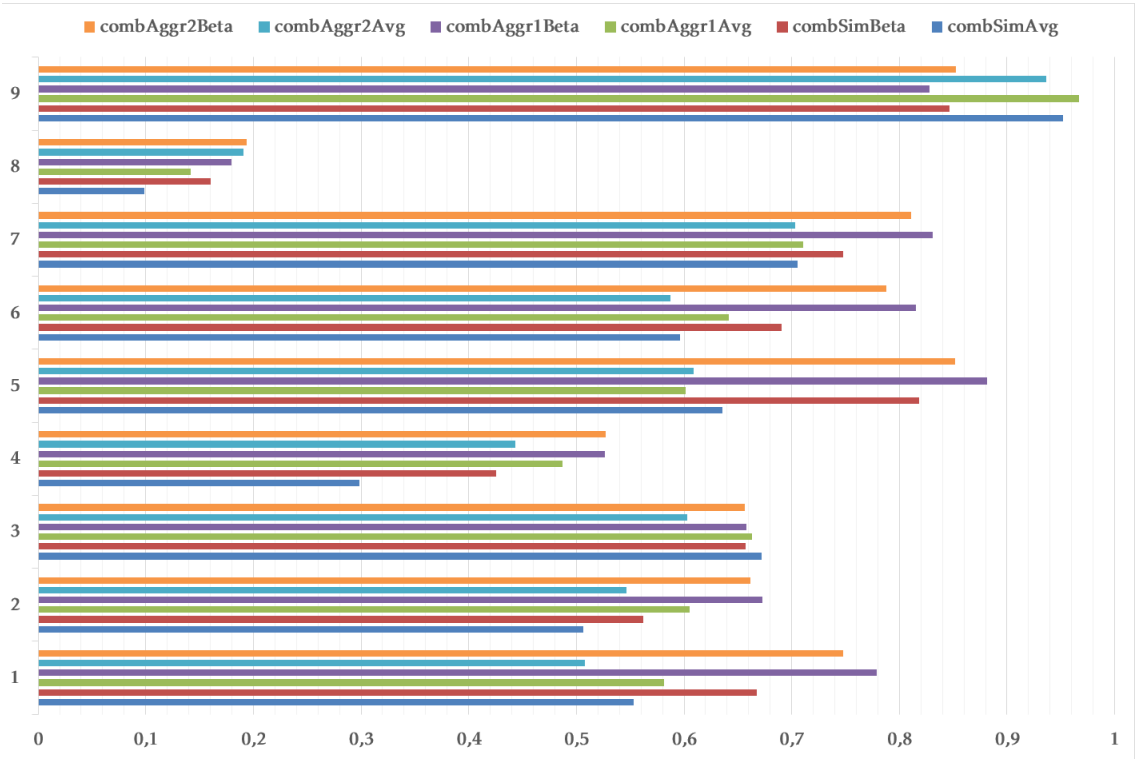


Figure 4.5: Combined arctangent and interpolation values for user ratings (simulation 1).

From the results presented above, it is possible to assert that the weighted averages (Aggr1Avg, Aggr2Avg, etc.) have inflated values, which is expected as they value more positive feedback elements such as comments. The beta distribution shows a different range of results from those given by the averages, which are exaggerated by the effect of the arctangent. In turn, the arctangent function was purposefully chosen as its slope quickly separates the ratings with positive values from those with negative values.

The scenarios previously presented about the users 3, 7 and 8 given the input feedback data can be observed in some of the rating values, mostly on the *combAggr2Avg* (light blue in Figure 4.5). The weighted averages with the beta distribution applied (*Aggr1Beta* and *Aggr2Beta*) have a higher value for user 3 than for user 7, and higher value than the remaining methods, with unexpected values like in the case of *combAggr2Beta* for user 4.

Despite the arc-tangent function being used to separate the positive and rating values visually, it lacks nuance to show which rating is better. Notice on the *atAggr1Avg*, in green, all users except 4 and 8 have this value between 80% and 90%. This effect is attenuated when combined with the values of linear interpolation, improving the differentiation between ratings that are best (user 7) and those that are good but should not be highlighted. The same analysis to the *atAggr1Avg* done on the *combAggr1Avg* value to all users except 4 and 8 now has a range of values between 55% and 75%.

Also the effect of time for *Aggr2* should be more subtle, its effects are noticeable in all the forms of presentation of the *Aggr1* and *Aggr2* values. Clearly, the weighted averages

are giving more positive values because they take into account many feedback elements such as comments and downloads as positive feedback, yet this is attenuated using the combination with interpolation.

4.5.5 Simulation run 2

A second run of the simulation was made, in order to compare results of the first data set presented and to use a slightly different weighted average from the one used in the first simulation. The new aggregation weighted average, *Aggr3*, values slightly more the negative feedback and also uses the age of the content to affect rating, but less than on *Aggr2*. These different feedback weights are called *WeightAlt* in the feedback types section (4.5.2). The data relevant to the simulation run is summarized in Table 4.3.

- User number **1** has very good rate on the votes and one fairly recent content element. Despite being only one content, its rating should be slightly positive, just above 50%.
- The rating of user number 1 should be lower than the rating of user number **7** (even if just slightly), which has more content elements. User number 7 has two reports and slightly positive votes, so the effect of the reports should bring the rating of the user down until they are dismissed or confirmed.
- User number **5** has only one content with mediocre votes, few votes and already over 100 days old. The rating for this user should be really close to 50%.
- Users number **6** and **8** both have mediocre or bad content and one report each; this should affect their rating. The fact that user 8 has more content elements should not bring its rating up by much, as feedback shows that only content 16 has some quality. Both their ratings should be below 50%.

In this simulation run the charts only differ from the first run with the addition of the *Aggr3* weighted average.

user id	content id	votes sum	#votes	#groups	comments	reports	#feedback	#views	age (days)
1	1	8	12	1	5	0	24	205	8
2	2	8	14	1	9	0	27	608	15
2	3	-3	11	3	12	0	31	355	96
3	4	9	17	5	4	0	30	423	79
4	5	6	24	1	5	0	35	630	83
5	6	1	19	2	6	0	30	244	104
6	7	1	19	1	9	1	35	703	93
7	8	5	13	3	9	0	25	312	37
7	9	7	25	3	11	1	45	350	102
7	10	-1	9	1	6	0	18	294	14
7	11	2	10	2	6	1	21	139	84
7	12	1	17	6	9	0	38	163	79
7	13	-2	14	2	4	0	23	482	42
7	14	1	13	1	11	0	28	385	100
8	15	-1	19	2	8	1	33	233	34
8	16	4	12	3	9	0	27	589	82
8	17	-4	14	7	6	0	30	127	83

Table 4.3: Content and user information for simulation 2.

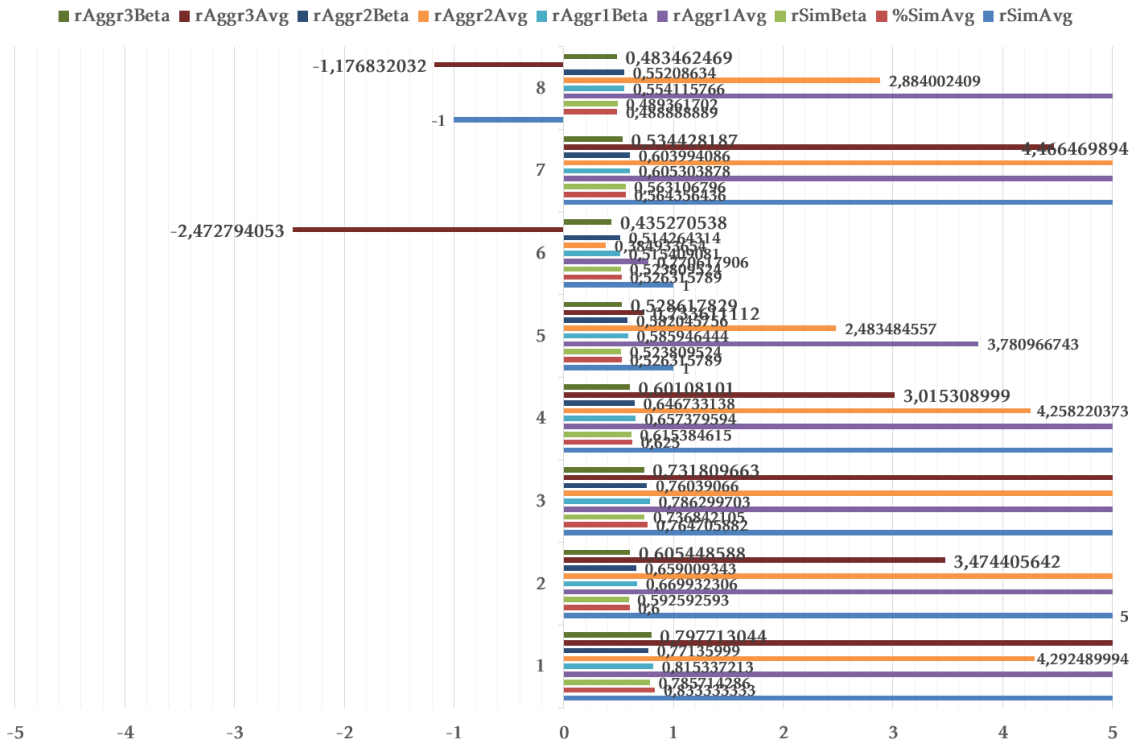


Figure 4.6: Raw values for user rating (simulation 2).

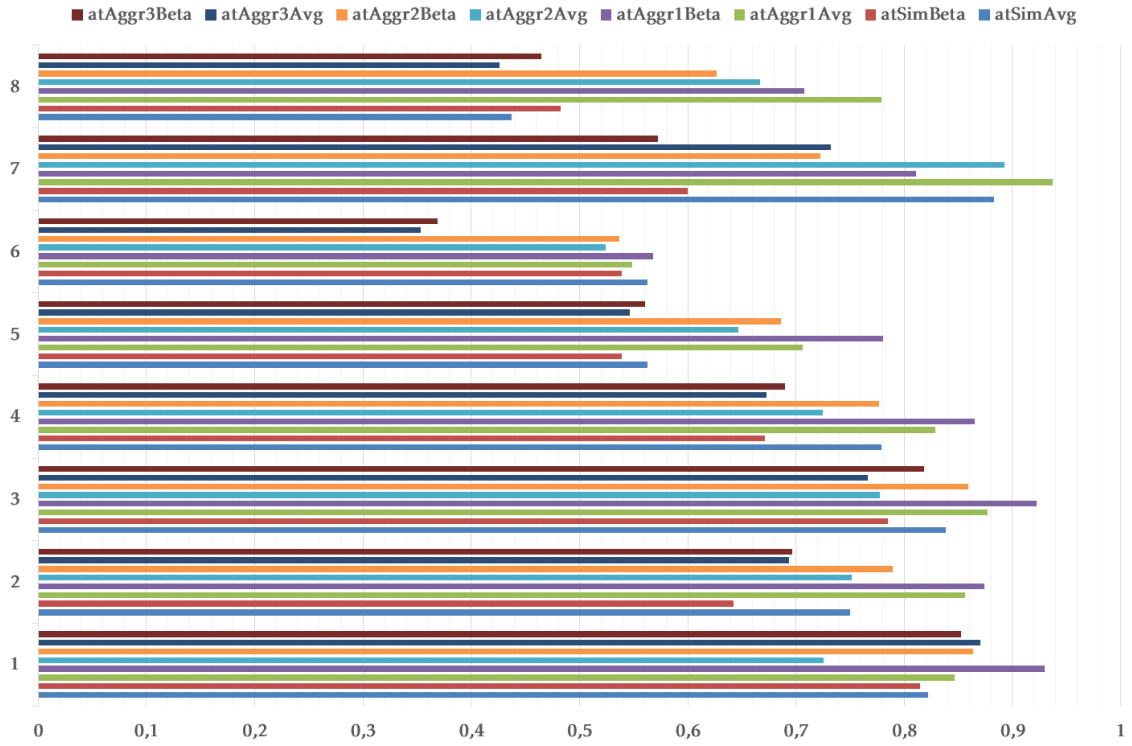


Figure 4.7: Arctangent values for user ratings (simulation 2).

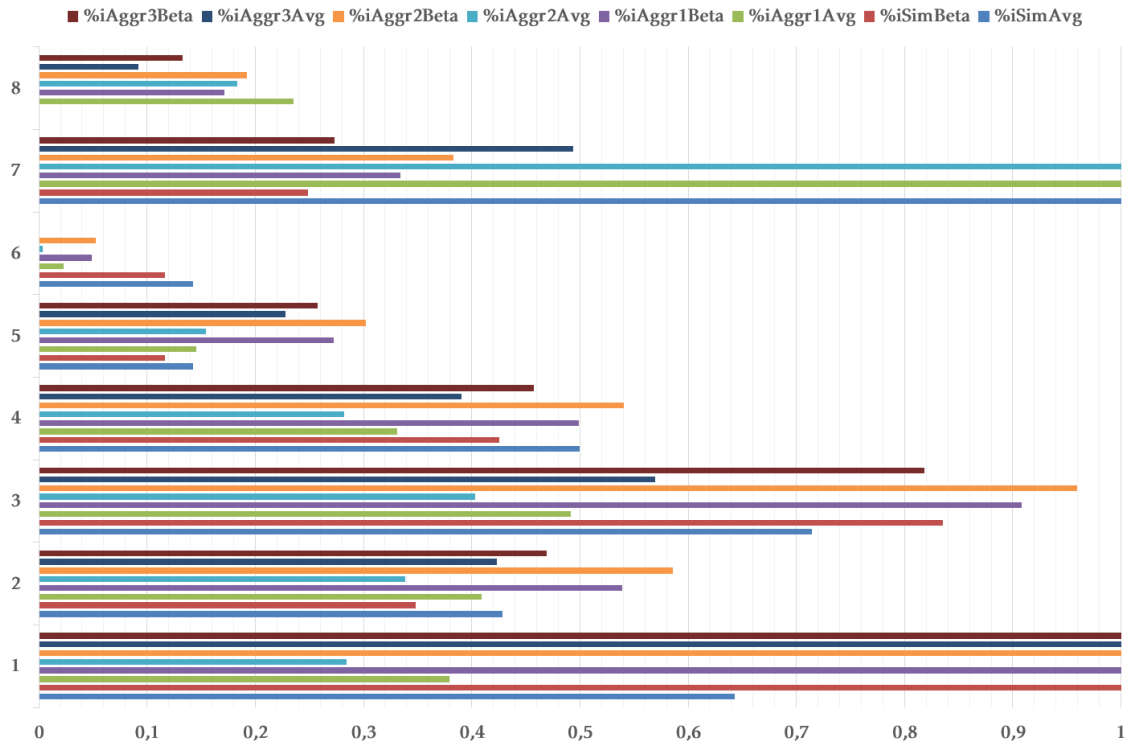


Figure 4.8: Linear interpolation values for user rating (simulation 2).

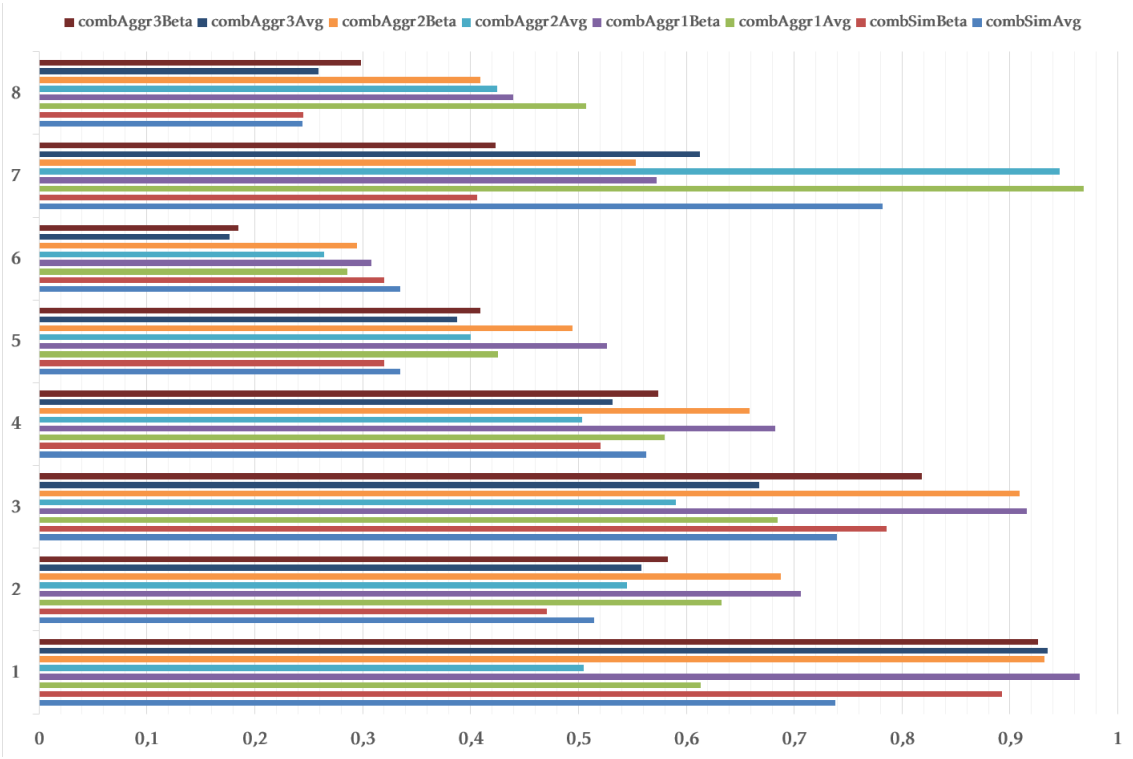


Figure 4.9: Combined arctangent and interpolation values for user ratings (simulation 2).

It is verified once again that using the beta distribution with the weighted average as input gives out a few unexpected results, which might be justified by this function not being appropriate for that usage. The most appropriate implementation of the probability distribution is only on the votes, where feedback does not use different weights, it is just either positive or negative unitary.

The first result that stands out is the high rating for user number 1. He/she has created only one recent content element with mostly positive votes. As the ratings are created by averages from the content, having only few almost entirely positive feedback elements is better than having many mostly positive feedback elements over multiple contents. This is a problem; the aggregation method for user rating should at least take into account the number of content elements the user has created.

The *Aggr4* weighted average is based on *Aggr3*, using the same alternative feedback weights, but taking into account the amount of content the user has created. Due to the bad results of using the beta distribution on the weighted averages, and also for clarity, these were removed.

With these modifications, going back to the case of user number 1 gives a result (*combAggr4Avg*) that is more reasonable. The second point made before the presentation of rating values, about user 7, now presents a rating value that is above the expected due to owning more content elements. From this data set, this user has the best rating on some methods, so the interpolation value is 100 %, which increases also the combined

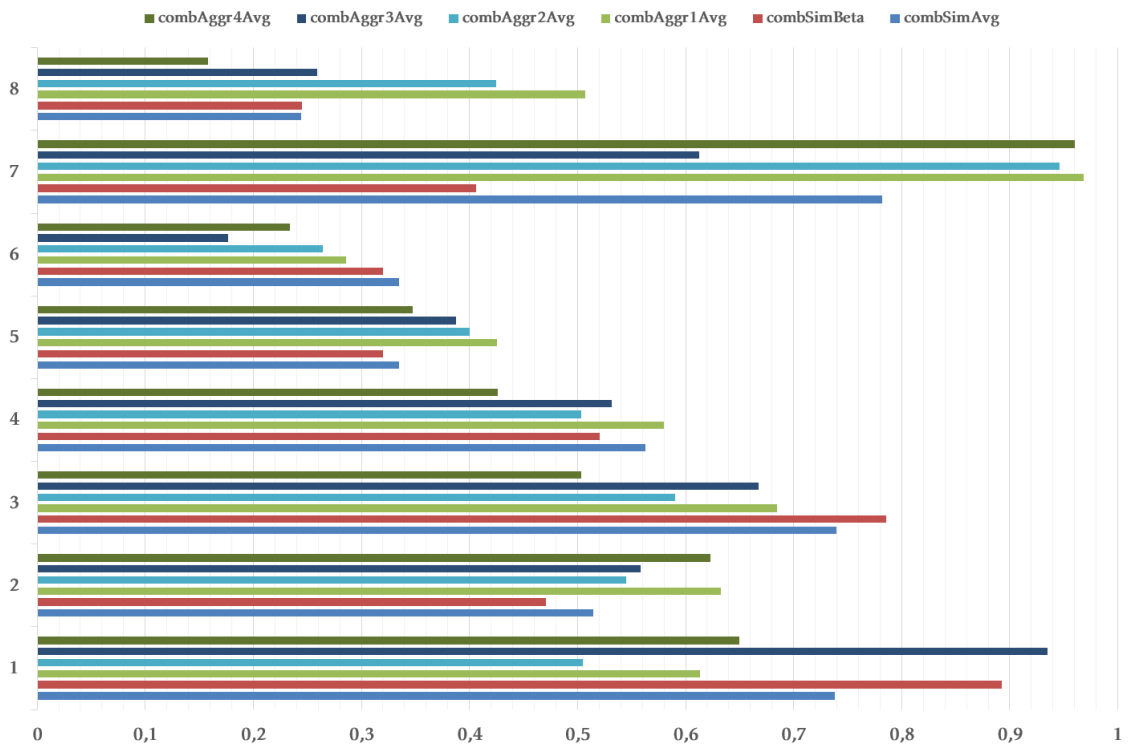


Figure 4.10: Improved algorithm (*Aggr4*) combined value simulation run 2.

result. With more users this would be mitigated, having users with better rating than this.

Continuing the analysis of the results, the **"combAggr4Avg"** has a descending trend from user number 1 to user number 8, excluding 7. This seems to be a reasonable result for these rating values, observing the data set information on table 4.3, yet user number 3 would not be expected to present a rating value below 50%. For user 8, the fact that the multiplying factor is greater due to owning more content elements aggravates its negative raw rating, making it lower than the one from user number 6.

4.5.6 Verifying methods on simulation run 1

In order to have a more complete analysis of the results from the new weighted average methods, these were also applied to the data of the first simulation run. Going back to this data set and applying the same methods as in Figure 4.10, we obtain the results presented in Figure 4.11. This verifies that the fourth weighted average method shows better results compared to the previous ones. It highlights the user number 9 with best rating value, and gives a slight penalty to users number 1 and 5, which only have one content element each with mixed feedback.

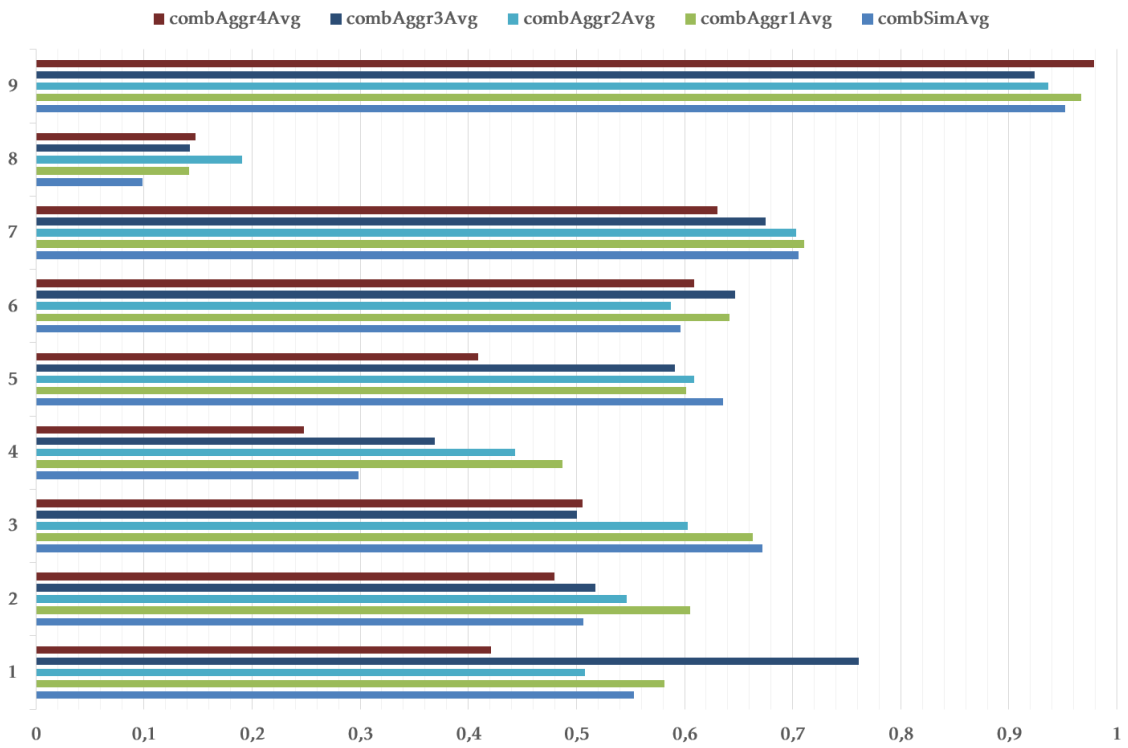


Figure 4.11: Using all algorithms with simulation run 1 data.

4.5.7 Simulation conclusions

Given the results for each method on the simulation runs, a conclusion can be made that using a larger set of feedback elements and other parameters for calculating the ratings can give much more meaningful rating values. This might have the downside of undervaluing votes, as bad content can have many comments and these are always considered as "positive" feedback. The "combined" rating value (*comb*) is good at showing the effect of relative quality between users and contents. This combination of the linear interpolation and the arc-tangent percentage values has the downside that new users and contents might not have 50 % rating value when they start using the platform. This can be an advantage to make users start creating content, yet it can be a penalty for very recently created content.

More research should be made to take into consideration other different weights and methods for using the feedback. Creating "ideal" weights for each feedback type automatically is not possible without having a way to measure the quality of the resulting rating value. This simulation effort could also be made to analyse the rating of each content element individually. With usage data collected from the system, another simulation could also be made using the observed dynamics of contents, groups and feedback elements creation.

The fourth version of the weighted average algorithm (*Aggr4Avg*), with the combination of linear interpolation and arc-tangent values (*comb*), is the best performing algorithm on the simulation. Therefore, the *combAggr4Avg* method is the one implemented on the rating system and described in detail by the Equations 4.6, 4.7 and 4.8. These are the content raw rating value, user raw rating value and percentage value (same equation for contents and users), respectively.

The counters for each feedback type on the content raw rating equation were abbreviated in order for it to fit the page. These abbreviations stand for: U (Up votes), D (Down votes), S (group Shares), C (Comments), R (Reports), V (Views).

$$ContentRaw = \frac{U*1 + D*(-1.1) + S*0.4 + C*0.05 + R*(-5) + \log_{10}(V+10)*0.1}{min(2, max(0.75, \log_{10}(1 + Time*1.7))} \tag{4.6}$$

$$UserRaw = \left[\sum_{content=1}^{n} ContentRaw\right] * max(0.1, min(3, \log_{10}((\#contents*0.7) + 0.01) + 0.3)) \tag{4.7}$$

$$Percentage = \frac{\frac{\arctan(RatingRaw)}{\pi} + 0.5 + \frac{RatingRaw - minRatingRaw}{maxRatingRaw - minRatingRaw}}{2} \tag{4.8}$$

Chapter 5

Tools and development approach

This chapter introduces a brief justification of the selected tools, their implications and setup. Furthermore, the approaches used in implementing the system, its architecture and design process are discussed.

5.1 System Structure

The web application is implemented using the *Play!* framework, a web development framework running on the JVM. This framework enables developers to write their applications both on Java and Scala languages, and provides an integrated web server, asynchronous response to requests to the server and management of front-end assets (JavaScript, CSS, files...). These features are particularly relevant for the identified requirements, and the reasons for choosing this framework. The data persistence is achieved using MongoDB, storing data as JSON documents. An overview of the project structure is shown in Figure 5.1 and the technology stack used to achieve it in Figure 5.2.

Several libraries and tools were used in the development process. The resulting files structure and routes/links structure are shown in the appendixes A and B. These give a clear picture of the application structure, scope and organization. Due to the complexity of the project, the count of fully implemented routes/URLs has gone up to 104 and source code files to 204.

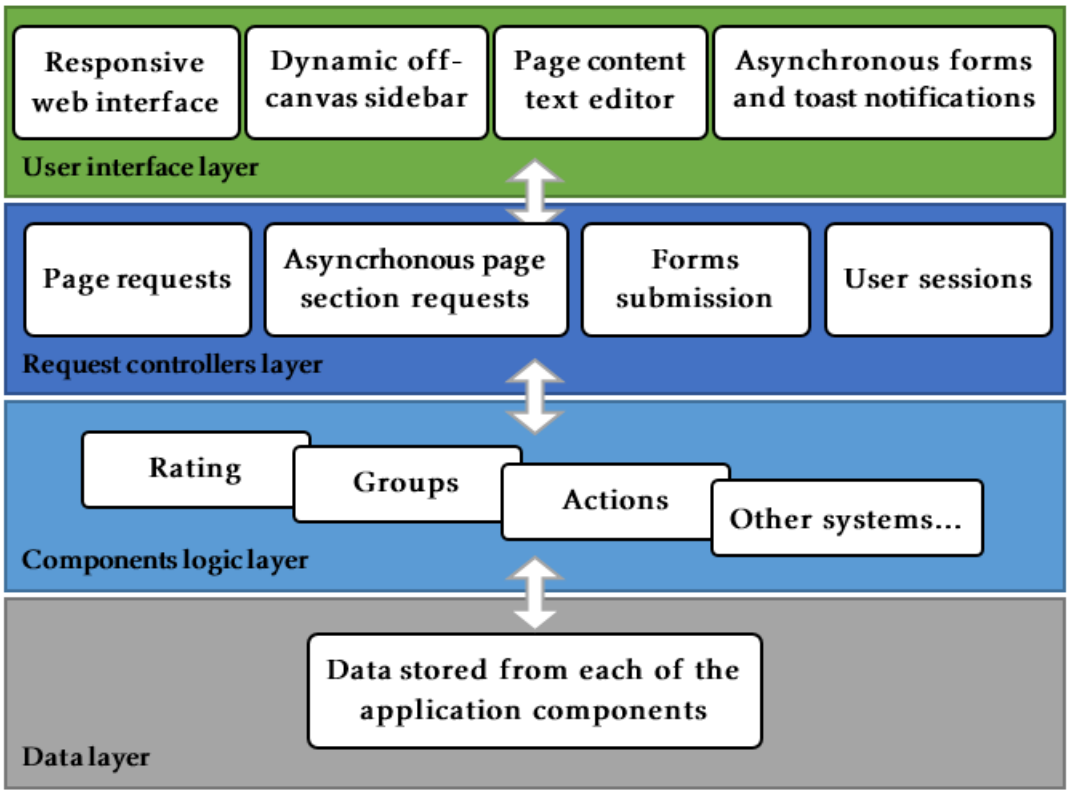


Figure 5.1: Overview of the system architecture.

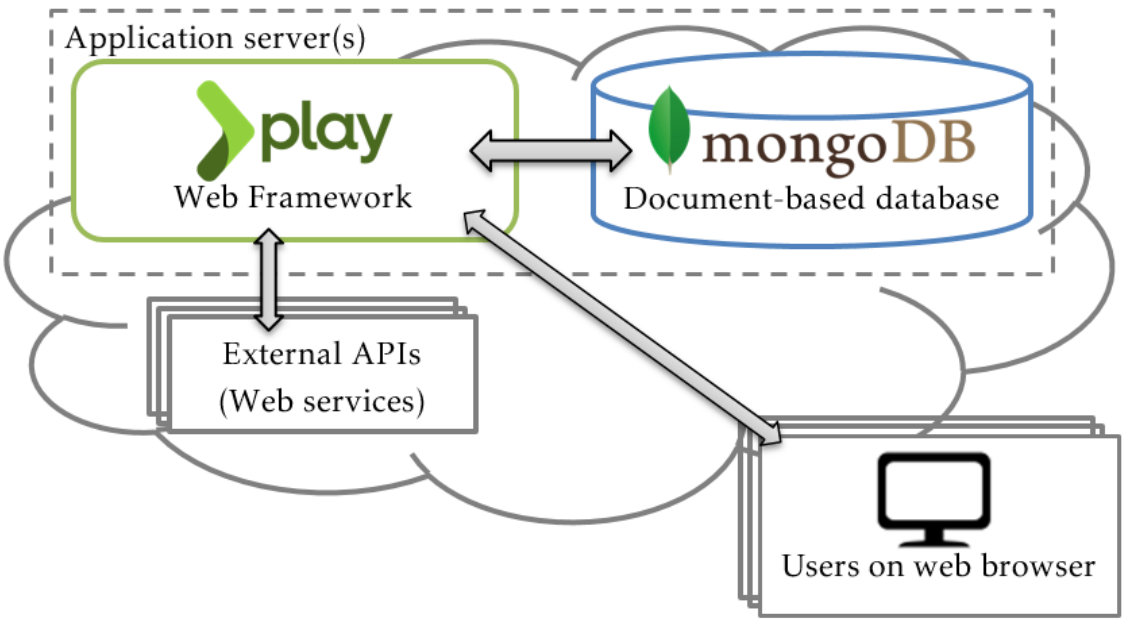


Figure 5.2: Overview of the adopted system technology stack.

Implemented features

The following list summarizes the implemented features of the system:

- **Authentication** system using strong encryption and password hashing, with sessions and cookies management.
- Safe mechanism for requesting password change, email change or for inviting users, with temporary **tokenized links**.
- **Asynchronous** web page behaviour based on AJAX and retractable sidebar.
- **Access control** system based on roles and context sensitive (as a platform user or as a member of groups).
- **Composite** or polymorphic data objects for tags system, access control, actions, groups and contents.
- Users **enrolment into groups** with one or more roles, such as member (generic), student, administrator, or professor.
- Creation of rich-text content pages using a What You See is What You Get (WYSIWYG) **editor** on the browser.
- Contents are rated through **feedback** given by users. Users are rated as well, through their created contents and actions. A visual cue is used for displaying these **ratings**.
- Integrated support for API development and external APIs usage, the routes system on Play framework is based on the REST structure, to ease implementation of web services.
- Built-in mailing system, send only.
- **Internationalization** (i18n) support, currently available in Portuguese and English languages.

5.2 Web Development Framework

In the beginning of the development, an HTML static prototype was created to illustrate the goals of this project (Figure 5.3). After prototyping statically the groups main page, a good framework allowing well organized development was searched for. Having already some of the needed utilities ready to help on the development, frameworks are mainly helpful in early stage development where it would be needed to create these tools first. Functionalities like serving dynamic templates that can be nested and re-utilized in different parts of the application, clear structure for receiving requests, accessing data, and serving the appropriate response, are readily available.

From the list of the previously discussed solutions for web development on the state of the art chapter (section 2.2.3), the Python framework Django seemed, at the beginning, an interesting and mature fit for the needs of the project. After fiddling around with the Django framework and its starter projects, the lack of familiarity with the language and strict structure of the platform made advancements on development difficult. Taking into consideration all the discussed web development frameworks or even the option of not using a framework and implement directly in PHP, the Play framework ended up being chosen. The documentation of this environment is very clear for starters, as well as the tutorial projects. As Java is a very familiar language, the development rapidly started.

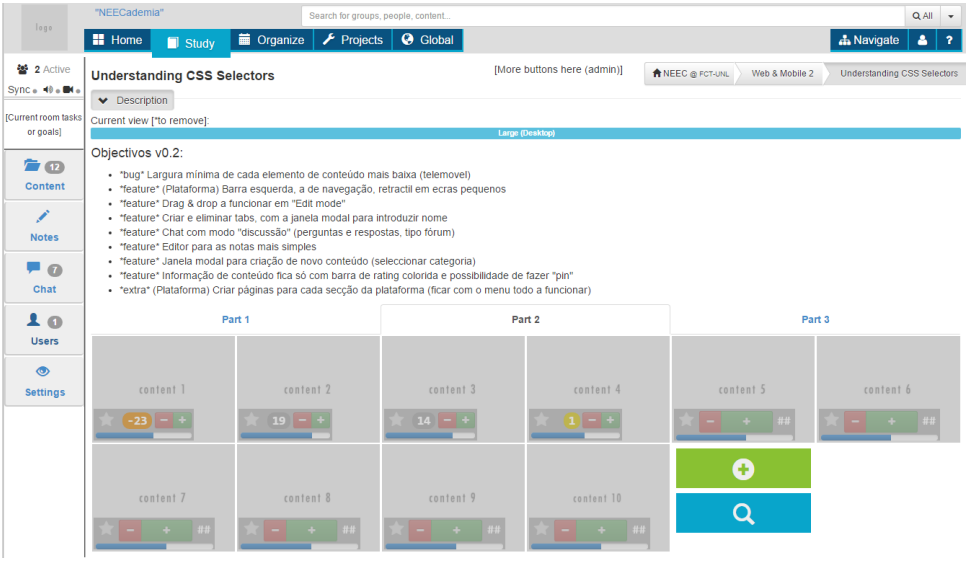


Figure 5.3: Early prototype of the website user interface.

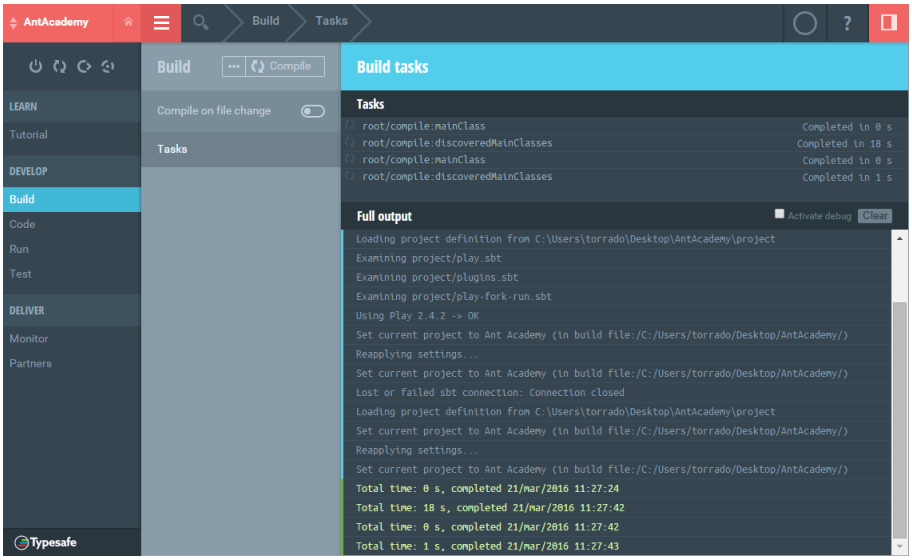


Figure 5.4: Activator graphical user interface.

Play uses the Scala Build Tool (SBT) for the build process, yet it also comes with a tool

called *Activator*, including a graphical user interface that runs on the web browser. This tool allows for compiling, running and even editing source code directly on the browser. This tools is shown in Figure 5.4.

An overview of the flow of requests made to the Play framework is shown in Figure 5.5 [47]. The diagram shown in this figure is related to version 1.0, but this MVC structure is still the same in the current version 2.4, used to implement the system. The framework has all the required functionalities provided in a well structured way and the documentation provides useful insights for developers new to the framework. Several starter projects are provided, ready to be run, exemplifying very well the basic features of the platform.

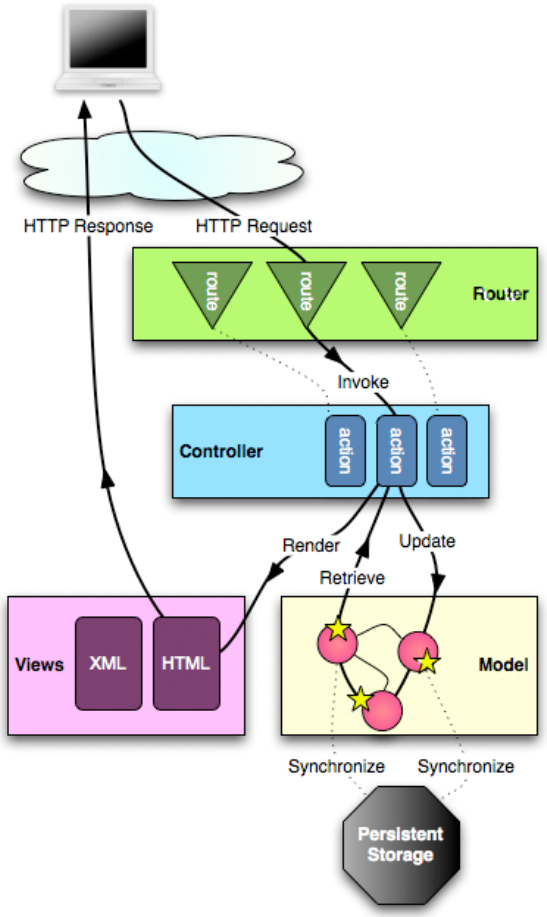


Figure 5.5: Play framework HTTP Request Path diagram.

The Play framework is based on asynchronous input output with dynamic routes. Many asynchronous features are implemented using Akka[1], a toolkit for developing message-driven concurrent applications on the JVM. Each time a request is sent, a *Future*

[1] http://akka.io/

is created, which is an Akka task created to asynchronously call the corresponding controller method. This also enables to create worker / background threads and scheduled events.

Both Akka, the Play framework and the Activator tool were developed by Lightbend (previously named Typesafe), a company heavily focused on the Scala programming language.

5.2.1 Model View Controller

As already pointed out, the Play framework is based on the MVC model, as it is the case with most web development frameworks. The system layer structure presented on figure 5.1 maps into the MVC design as:

- Models include the data layer and part of the components logic layer, with queries and basic data operations.
- Views are part of the user interface layer, in combination with assets like CSS, JavaScript and font files.
- Controllers encompass the request controllers layer, as well as the remaining part of the components logic layer not included in the models.

A big part of the views is using the templates system of the framework, which is based on Scala and allows for embedding Scala source code in the templates. This is specially useful for calling Java functions in the project. Another feature in the templates that is very useful is the syntax for creating conditions and loops. The framework also features automatically generated helper classes with the reverse routes, giving the URL related to a given controller method and its parameters, which is very common in MVC frameworks.

5.2.2 Used Libraries

Several libraries or frameworks were used in the development of the application, besides the libraries included as part of the Play framework itself. These libraries are listed below, separated by programming language.

5.2.2.1 CSS

Libraries used for CSS:

- Bootstrap [2] – This front-end design framework provides several stylised design building blocks for web user interfaces such as buttons, lists, drop-down menus, responsive grid system, to name a few. It uses JavaScript as well, but it is also more concerned with the dynamics of presentation in these components (like in

[2] http://getbootstrap.com/

the case of transitions). It is open-source and also includes a set of font icons from Glyphicon.

- Font Awesome [3] – Large open-source collection of easy to use font icons.

5.2.2.2 JavaScript

Javascript libraries used:

- jQuery [4] – The most popular JavaScript library. *"It makes things like HTML document traversal and manipulation, event handling, animation, and Ajax much simpler with an easy-to-use API that works across a multitude of browsers"* [4].
- Toastr [5] – jQuery based simple toast notifications library (small balloon notifications at the bottom right corner of the page).
- TinyMCE [6] – *"a platform independent web-based JavaScript HTML WYSIWYG editor control released as open source under LGPL"* [6]. Used for editing content pages.
- DataTables [7] – jQuery plugin for building advanced tables with interaction controls (ordering, pagination, etc.).
- Clamp.js [8] – cuts off text that is too big to fit in its container and adds an ellipsis to it (for example, "text too long for ...").
- Modernizr [9] – library that eases checking if the user's browser has certain HTML, CSS or JavaScript features.

5.2.2.3 Java

Java libraries used in conjunction with the Play Framework:

- jBCrypt [10] – library with implementation of the improved Blowfish password hashing code, called bcrypt. Further discussion about this item on section 5.2.3.
- play-jongo [11] – Play module for Jongo [12], an Object-Document Mapping (ODM) library for MongoDB, mapping database documents into Java objects and vice-versa. It also features a query syntax very similar to the one used in the MongoDB shell.

[3] http://fontawesome.io/
[4] http://jquery.com/
[5] https://codeseven.github.io/toastr/
[6] https://www.tinymce.com/
[7] http://www.datatables.net/
[8] https://github.com/josephschmitt/Clamp.js/
[9] https://modernizr.com/
[10] http://www.mindrot.org/projects/jBCrypt/
[11] https://github.com/alexanderjarvis/play-jongo
[12] http://jongo.org/

- play-mailer [13] – Official Play framework email module.

5.2.3 Cryptographically secure password hashing

This section details the choice of the password hashing algorithm, as well as its usage. The bcrypt algorithm is used by several Linux/Unix distributions like BSD and SUSE Linux [48] as the default password hash algorithm, replacing other commonly used password hashing algorithms like MD5 and SHA1, both found to be vulnerable [49]. SHA1 will no longer be used for HTTPS due to its vulnerabilities, taking place during this year (2016) a transition to SHA2 (or better) [50].

$$hash = bcrypt(password + salt) \tag{5.1}$$

Bcrypt also creates a secure salt, random data appended to the user password, so that the hashing algorithm input is different even if multiple users have the same password. With the salt, the hashed value cannot be directly correlated with the password, avoiding attacks like rainbow tables. Rainbow table attacks use large lists of pre-computed hash values for commonly used passwords, to know what the input password was.

5.3 Database

The requirements on the data models are quite complex [14], with special emphasis on the need of extensive polymorphism in aspects of the platform such as the contents, groups and access control. The database design started out sketched as a 39 tables entity relationship diagram (figure 5.6) to be implemented with MySQL, as SQL was the technology most utilized in other college-related and extra-curricular activities. The entity relationship diagrams have been developed using the already mentioned MySQL Workbench tool, that allows forward-engineering and backward-engineering between the diagrams and SQL data definition language code.

The Play framework is integrated with the Ebean Object-Relational Mapping (ORM) tool, which allows for the developer to create the data models as Java classes (or Scala) and lets the framework generate the appropriate database creation or evolution code. SQL data definition language is generated and ran automatically, as well as managing the conversion of query results into objects, instances of those Java classes, the models.

After having several models, their variety and respective queries became increasingly complex and hard to maintain. The data structure might have been a more denormalized design, using less but more complex data tables, which could mitigate some of the encountered difficulties. For easier management of complex queries to aggregate data, an alternative to the relational database solution was prospected. The data design was

[13] `https://github.com/playframework/play-mailer`

[14] This project does not go into details concerning scalability, which would bring new challenges and approaches to database design, using database replication, sharding and clustering.

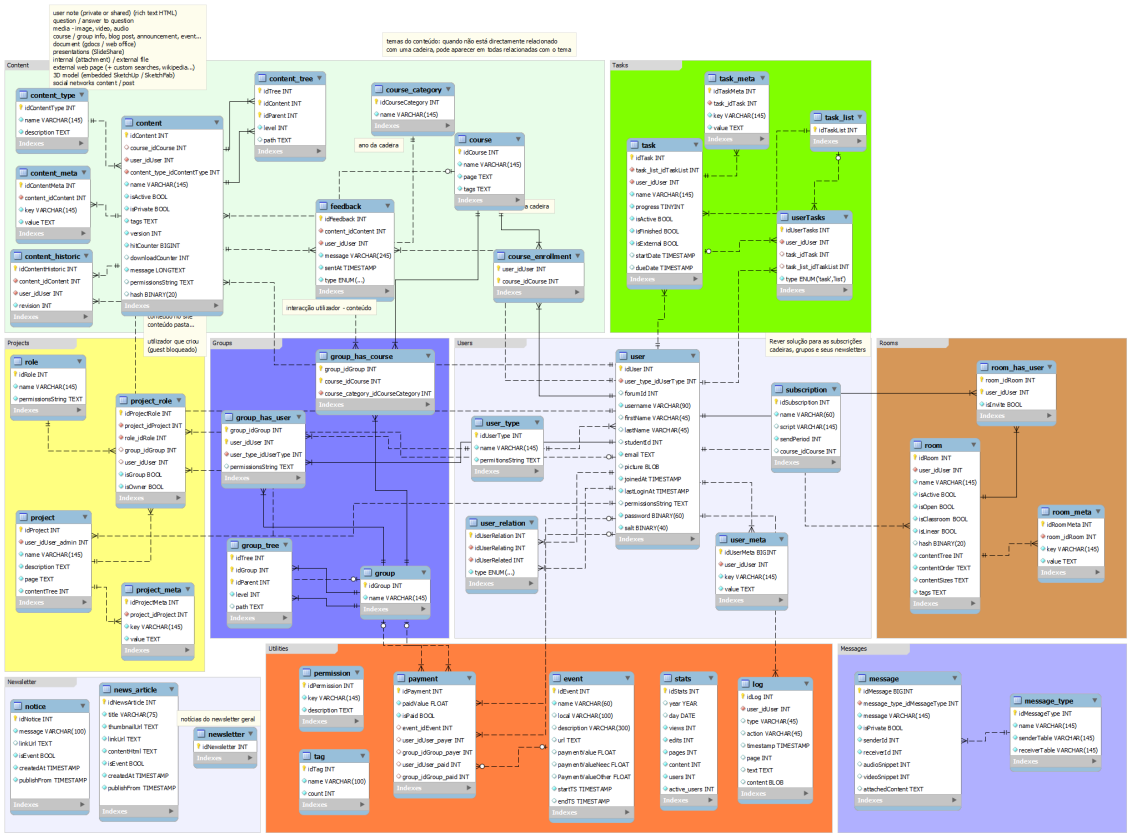


Figure 5.6: Early database sketch on SQL, final MongoDB scheme on Figure 6.1.

appropriately fit to be implemented using a document-oriented database, as there is a great focus on the content and group objects and these contain many different elements inside them and are very unstructured.

The database technology chosen to address the requirements and facilitate development was the MongoDB database solution, a document-oriented database with a strong community and accessible entry-point. It had a mild learning curve while providing the expected features. In comparison with SQL databases, document-oriented databases store a *collection* of *documents*, making documents the equivalent of SQL rows and *collections* the equivalent of tables. The documents are stored in a format similar to JSON called BSON, a binary representation of JSON documents used by MongoDB.

During the project development, graph-oriented databases were also considered, as these would provide a great way to create a graph for the interconnected users, groups and contents. This would be a clever, but excessively contrived way of depicting the tree-like simple hierarchies that ended up being designed for the project. With no constraints to the project it would be in itself a great subject of research and development (and it is, as previously discussed in section 2.2.4), bringing new capabilities to exploration and organization of contents, groups and users on the platform. The tree structure for groups is possible to implement in both SQL and MongoDB, with appropriate tree structure data

designs, and these are present on both data designs, even if it is a functionality that ended up not fully implemented.

5.3.1 Structuring data in MongoDB

The data in MongoDB has no *schema*, so unlike SQL data objects, these do not follow a strict data structure, which makes it harder to draw a detailed diagram of the data model and makes the design process of the database slightly different. This does not mean that documents are not structured and designed using several patterns. As in SQL, there are choices regarding normalization of the data model and granularity of data regarding read or write performance and query complexity for the application's purposes. Here operations are more focused on the individual collections rather than on relational databases, and therefore must take into consideration the *atomicity* of data. The database operations are atomic at the document level, so it has to be decided if a data model embeds related data or if references are stored between related pieces of data. So, having data as an embedded document reduces the ease of querying it, as it is dependent on the parent document, and referencing other document with the data makes the need for doing another query to get data after getting the reference.

As previously stated, MongoDB documents are stored in the BSON format, including the data types from the JSON format:

- Number, a signed decimal number, no distinction between integer and floating-point, but implementations of the format can encode them differently;
- String, sequence of zero or more Unicode characters;
- Boolean, `true` or `false` values;
- Null, an empty value;
- Object, an unordered collection of key-value pairs:
  `{name: "an object", embedded_object: {value: 2}};`
- Array, ordered list of zero or more values: `[2, "hey there!", [null]];`

These data types are extended in BSON, with further data types like: raw binary data, ObjectId, Date, DBPointer, Regular Expression, Symbol, Timestamp and JavaScript. These types enable creating richly composed data documents and they are all comparable and sortable by the database, while being able to be serialized back into standard JSON.

The data embedded in documents should be considered according to what data needs to be retrieved by the application. For example, a group document should at least have all the data necessary to open the main page relating to this group. Ideally, the goal is for reducing the number of queries for opening pages down to just one.

5.3.2 Referring to objects

By default, the MongoDB database creates an unique identifier, an *ObjectId*, which is stored in the unique field of the collection named *_id*, which acts as the primary key of the collection. The *ObjectId* is a 12-byte identifier that is constructed as follows [51]:

- 4-byte timestamp, representing the seconds since the Unix epoch;
- 3-byte machine identifier;
- 2-byte process id;
- 3-byte counter, starting at a random value.

With the timestamp at the start of the the *ObjectId*, it enables access to the creation time of the object on the database and sorting these values is roughly equivalent to sorting by creation time. Usually, database documents are created without the *_id* field and it is automatically created by the database with an *ObjectId* value. If the *_id* field has a value when storing a new document on the database it replaces the default *ObjectId* value, yet this still needs to be a unique value.

One requirement set out to achieve on the project was the ability to link together different types of objects from the platform. One example of this is the actions system, where the *actor* and *context* have a fixed type, **User** and **Group** respectively, yet the *target* and *data* objects can be of any of the major data structure types. The *ObjectId* in conjunction with the data *collection* name define both the object type (created the *ObjectType* class) and identification of a reference.

5.3.3 Using MongoDB

MongoDB installation is very simple. To get a database server instance up and running we just need to extract the database program files and on the binaries run *mongod*, the database server process. It is highly portable, with low system requirements and many tools and plugins available to work with it.

As already pointed out, the application uses the the ODM Play-Jongo, Jongo plugin implementation as a module for the Play framework. This enables mapping Java classes to MongoDB database collections and querying the database. The only slight down side of Jongo is relying on a single string for parameters for each query operation (*find* is one operation). This becomes an issue when there are many parameters and strings become long and confusing. Other implementations, like the PHP Fat-Free Framework, instead use an approach where each parameter is delivered inside arrays, or even based on functions/methods that construct the string. A hypothetical example with functions in a Java class called *Params* is shown in Listing 5.1).

Listing 5.1: PlayJongo example and critique

```
1 //usage example of PlayJongo / Jongo
2 friends.find("{age: {$gt: #}}", 18).as(Friend.class)
3
4 //should use some kind of structure not hard-coded
5 friends.find(Params.list(Params.create("age", greaterThan(18))).as(Friend.
      class)
```

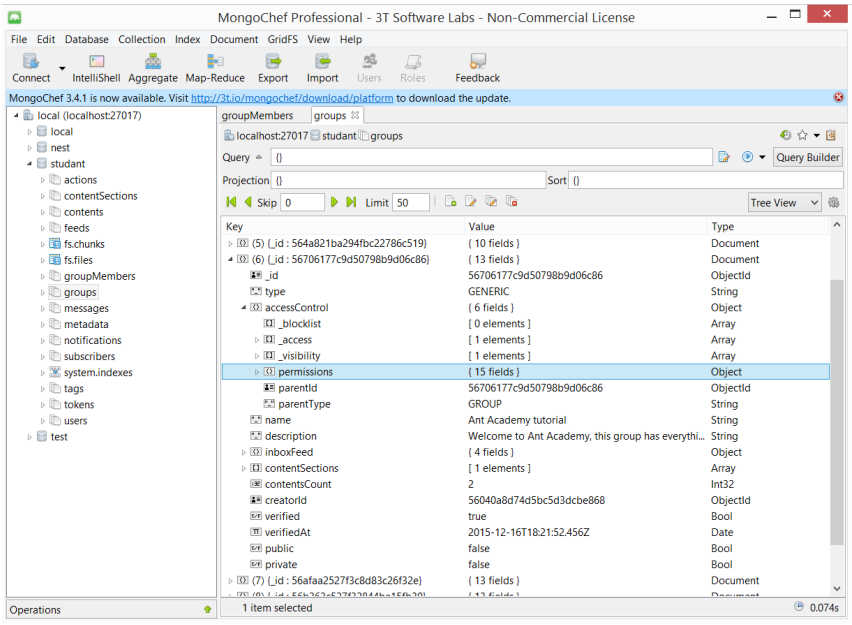


Figure 5.7: MongoChef graphical user interface.

To manage the MongoDB databases, the software MongoChef[15] was used. The software provides several utilities to manage and visualize data, as well as helpers for building queries. This makes Jongo and MongoChef more useful together as it enables for easier creation and testing of queries on MongoChef and passing these into code using Jongo. The user interface for data navigation as a tree structure is shown in Figure 5.7. MongoDB also includes a command line tool for this purpose, yet using a graphical interface is much more user-friendly and still enables to write the same commands.

For querying the database collections, there are the basic CRUD operations, the aggregation pipeline and map-reduce operation. The queries, like the documents, use the JSON format. The CRUD read operation is the "find" used in Listing 5.1, where the query string contains the field being searched and either the intended value or an operation (in that case "greater than 18").

The aggregation pipeline is a MongoDB framework for aggregation of data in a multi-stage pipeline. It enables for complex queries, mainly on grouping documents. Map-reduce is a data processing paradigm to aggregate large volumes of data, using JavaScript

[15] http://3t.io/mongochef/

for the map and reduce phases functions. Map returns one or more objects for each input document, while reduce combines the results of the map function. *"While the custom JavaScript provide great flexibility compared to the aggregation pipeline, in general, map-reduce is less efficient and more complex than the aggregation pipeline"* [52].

Lastly there is the GridFS, a file storage feature in MongoDB databases. This stores files on the database, listing them on the *fs.files* collection. MongoDB has a 16 megabytes document size limit, files bigger than that are stored in chunks on the *fs.chunks* collection.

Chapter 6

Implementation

This chapter focuses on the details of implementation of the system. As already pointed out, the architecture of the web application is divided into four different layers: the data layer, the components logic layer, request controllers layer, and user interface layer. Each of these layers is presented here, starting with the data, followed by the user interface, handling requests and then the implementation of the main and secondary components.

6.1 Data Layer

This section describes the data structures and modelling implemented for the system. After having the web application user interface and systems sketched out, the organization of data should follow the needs for those specifications. The following diagram (Figure 6.1) shows the MongoDB collections used in the project database and the main references between these collections.

Green collections are the data models for the main component of the application: **"users", "groups" and "contents"**. These collections always have an **AccessControl** object embedded as well as other related objects (or part of them). This speeds up opening the pages related to these items, reducing the number of database read queries (ideally to 1), making it unnecessary to read the associated documents to show the page related to that object. In Listing 6.1, an example of a document from the "groups" collection is shown. It shows that the feed of notifications, as well as the list of content sections and their contents, are embedded in the document. Yet, those objects also exist in their own collections, this is just a copy, like a cached or materialized view in SQL.

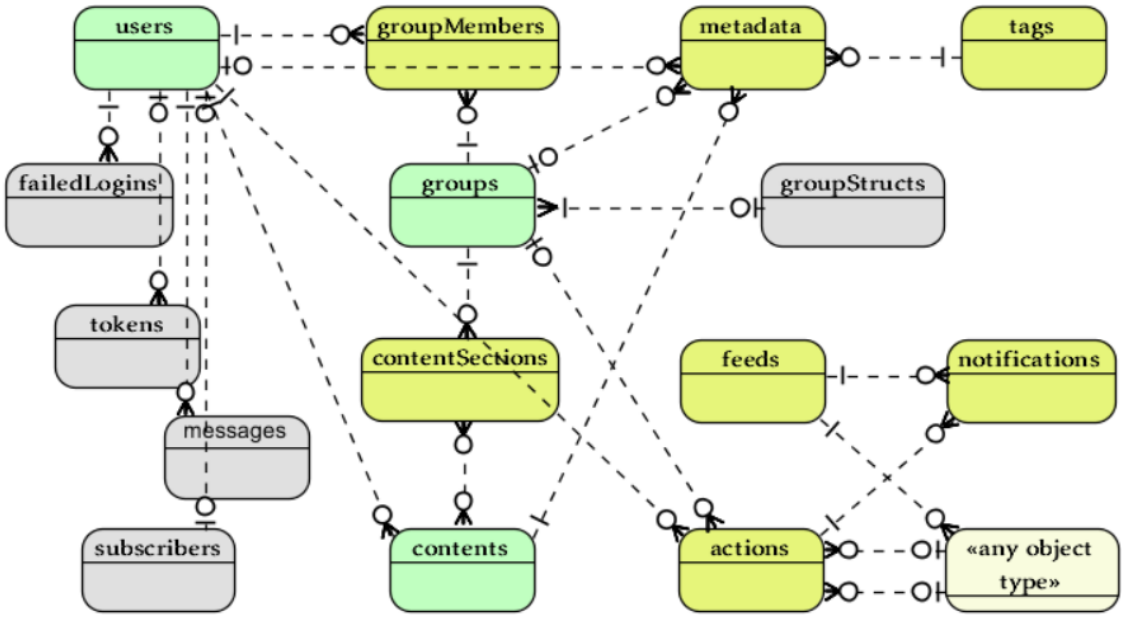


Figure 6.1: Conceptual data model: entity-relationship diagram for MongoDB data.

Listing 6.1: Group data model - simplified JSON example

```
{   "_id" : ObjectId("56706177c9d50798b9d06c86"),
    "type" : "GENERIC",
    "accessControl" : { "_blocklist" : [ ],
        "_access" : [{"domain" : "GLOBAL", "type" : "G_ANY"}],
        "_visibility" : [{"domain" : "GLOBAL", "type" : "G_ANY"}],
        "permissions" : { "read_chat" : [
                {"domain" : "CONTAINER", "type" : "C_GROUP",
                "containerId" : ObjectId("56706177c9d50798b9d06c86")} ],
            "write_membersRequests" : [...],  ...
        }, ... },
    "name" : "Ant Academy tutorial",
    "description" : "Welcome to Ant Academy, this group has everything you
       need to be a pro using this platform!",
    "inboxFeed" : {
        "id" : ObjectId("56706177c9d50798b9d06c87"),
        "type" : "FEED", "value" : NumberInt(0), "other" : {} },
    "contentSections" : [
        { "_id" : ObjectId("56706177c9d50798b9d06c88"), "name": "default",
            "groupId" : ObjectId("56706177c9d50798b9d06c86"),
            "contents" : [ ObjectId("565871ea6152d3d01fd78346"), ObjectId("560
               fed38f20d2e74a4cddb14") ] } ],
    "creatorId" : ObjectId("56040a8d74d5bc5d3dcbe868"),
    "contentsCount" : NumberInt(2), "verified" : true, ... }
```

The collections in yellow are related to data accompanying the main components and with sub-systems. The tags sub-system data is stored on **"metadata" and "tags"**, actions and notification feeds are stored on **"actions", "notifications" and "feeds"**, while **"groupMembers" and "contentSections"** store the users and contents related to each group, respectively.

Collections in grey are the least relevant for the application system, these are:

- "failedLogins", stores instances of failed login attempts to limit retries, avoiding brute-force attempts to find passwords;
- "tokens", a store of cryptographically secure tokens for purposes of user session, confirmation emails and other secure links that may or may not have an expiration limit or single use;
- "messages", hold text exchanged messages between users and messages sent to group chat;
- "subscribers" is just a collection of emails of people that want to receive newsletter emails regarding the project;
- "groupStructs" stores data defining a tree structure of groups as the array of materialized paths of the tree.

The data structure for representation of the group structures inside "groupStructs" is a tree representation called the *materialized paths* pattern. This design pattern is explained in the MongoDB documentation [52]. It stores the tree nodes paths as strings, so if we have a structure with root "Root", branch named "NodeX" and branch of "NodeX" named "NodeY", `",Root,NodeX,NodeY,"` is the path for the *NodeY* on the tree. In our case the path has the *ObjectId* identifiers for the groups on the tree, with the main group of the structure as the root. The pattern provides flexibility in querying tree paths using regular expressions, allowing to work with partial paths, but suffers in performance in cases of trees with greater depth.

6.2 User Interface Layer

The application user interface, the layout of the pages, and the way these are organized, are discussed in this section. The user interface of the web application is split into three parts/sections:

- Header, indicates the currently open section of the application (users, groups, etc.) and allows for quick navigation to these sections;
- Sidebar is retractable and dynamic, it shows helper tools like the application's chat;
- Main section, where the page's content is shown and determines the state of the header and sidebar, as the sidebar changes according to the open group or content.

6.2.1 Sidebar

One distinct feature of the website layout is the off-canvas sidebar. By default, the sidebar is hidden off-canvas, outside the scope of the page drawn on the window, and when shown it slides into view. The creation of the sidebar comes from the original sketches for the website. Its purpose is to allow users to be on a group page, with the grid of contents, opening or editing content while being able to write notes, chat or do other complementary tasks on the sidebar at the same time.

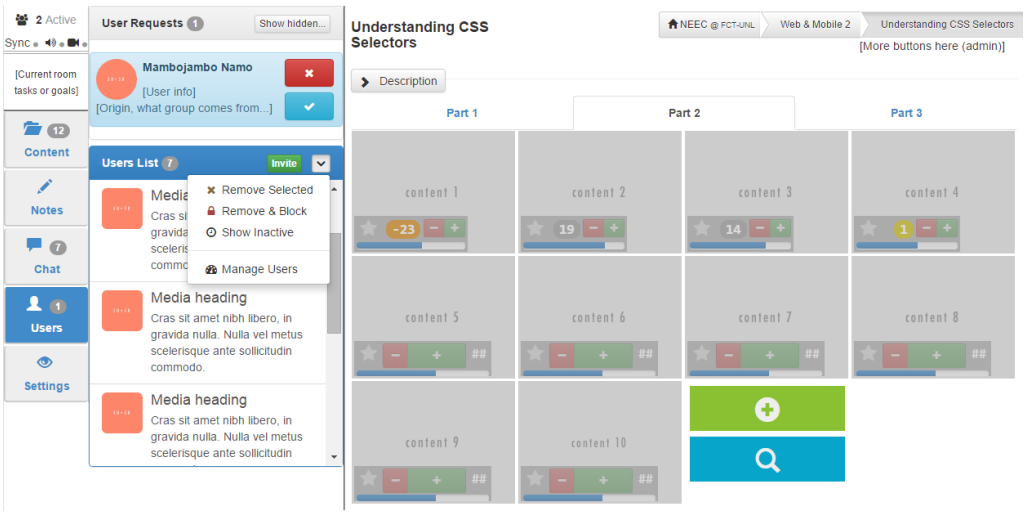


Figure 6.2: Sidebar on page sketch / prototype, similar to the final implementation.

This sidebar is fully Responsive design. When the screen is wide enough it appears besides the main page section, while on smaller screens the sidebar pushes most of the main section off-canvas to make room for the sidebar, in which case clicking on the main section will automatically close the sidebar to ease navigation. To have multiple tools on the sidebar it uses different panels for each of these tools interfaces, showing the appropriate panel, if selected, on the sidebar. On the panel selection buttons, labels with a number notify that there are new elements in that section since it was last opened. Figure 6.2 shows an early version of the sidebar, while Figures 6.3 and 6.4 show the final version of the sidebar. Furthermore, the sidebar reflects which group or content is currently open on the main section.

6.2.2 Responsive design

The Bootstrap front-end design framework features a responsive grid system, a grid of 12 columns to arrange interface elements horizontally. This enables to set different arrangement of user interface elements for 4 different screen widths (extra small, small, medium and large). The large screen size is shown on figure 6.3 and the small screen size on figure 6.4. When going to small sized screens, the sidebar uses a larger portion of the

width available and also the text from the header buttons is removed to make room for the buttons. Other page elements like forms, lists and the content grids are also responsive.

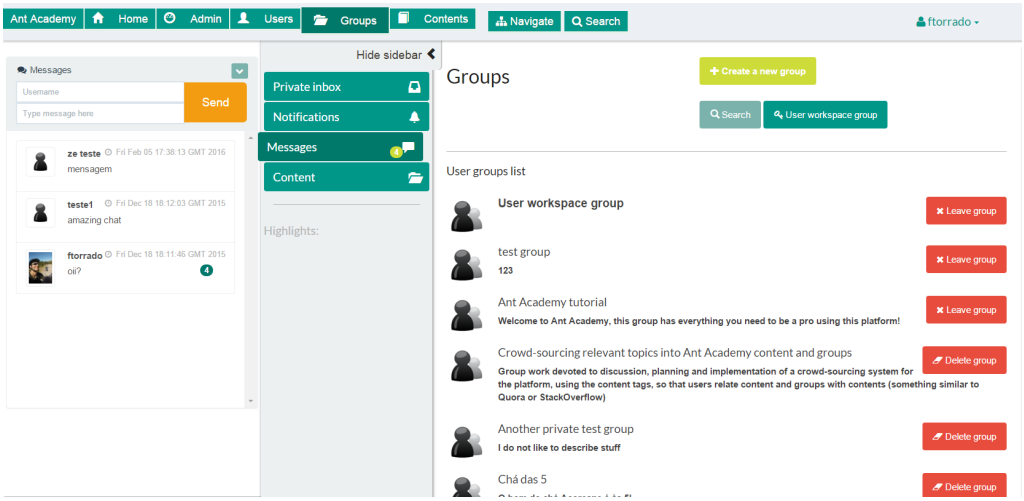


Figure 6.3: Page with sidebar open on wide screen.

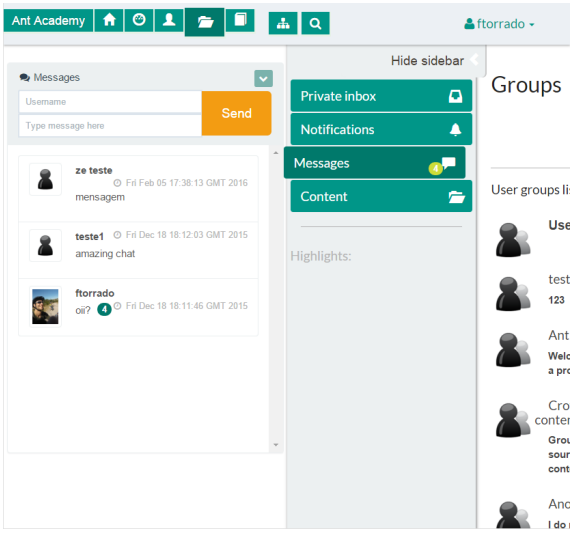


Figure 6.4: Page with sidebar open on narrow screen.

6.2.3 Asynchronous content

Web application are sometimes implemented as single page applications, which never refresh the whole web page, only updating parts of it, as described in 2.1. This approach is used for a variety of reasons, both aesthetic and functional; the most important are:

1. The ability to work on different parts of the page without losing information when another part of the page needs to change;

2. To not show the blank web browser every time the page changes and keep similarity with the behaviour of other computer applications.

Some single page web applications approach navigation between pages using a routes scheme based on the URL hash and implements routing on the front-end. One example of URL hash routing can be seen on Gmail: `https://mail.google.com/mail/u/0/#inbox/[mail ID]` loads the page for the email account (`https://mail.google.com/mail/u/0/`) and the URL hash (`#inbox/[mail ID]`) is used by the page to open the corresponding *inbox* message. This implementation only uses some functionalities of that concept or in some cases, as most of the times it does not refresh the whole page and keeps the same interface structure throughout the application. Keeping the structure of URLs coherent with the REST model, keeping the separation between application and client and not relying on JavaScript for presentation of the application pages (except more dynamic elements like the sidebar). These were the approaches taken while developing the graphical user interface.

The user interface of the application has 4 parts that can be asynchronously updated. These relate to the 3 already mentioned main sections of the user interface layout and are the main content section, the header (this one is never updated as it does not change), the sidebar and the panel inside the sidebar. Links on the application can have attributes that trigger the events of loading the requested page on that section or refreshing certain sections.

User data input like sending chat messages, commenting content or voting on content, is done in forms that should be asynchronous, because these are actions where the user should not be redirected to another page, which is the default behaviour of HTML forms. Some forms where there is no need to stay in the current page, like the form for creating a new content, are not asynchronous. These forms should be asynchronous too, yet with some extra logic in JavaScript to handle redirection of the page showing on main section. The links that open pages (on the main interface section) asynchronously have an indication in their HTML as the "async-content" class, and also have the "data-section" attribute for the header to show which website section is active.

6.2.4 Error pages and notifications

A set of error pages was created to accommodate several error situations. The *Play* framework has an easy way to return a response with the proper HTTP status code, which is great to create APIs and to indicate errors there. Yet, when some web browsers receive error status codes as response they will still show the content sent on the response body, so this does not indicate the error. To provide this information on the web browser, several error pages were created indicating what is happening: no permission (with notice if not logged in), not found, failure by bad request and failure by internal server error.

As the forms on the website are asynchronous, these may return multiple results messages of success, error or warning. The messages returned should be shown on the interface, so the Javascript library *toastr* is used to show notifications in the bottom right

corner of the page, with different colors to indicate if the message is of success, information, error or warning. These error pages and notifications can be seen on section 7.

6.2.5 Text editor and other plugins

Other major user interface structures used on the system:

- Grid of content elements (section 7.6), showing content thumbnail, rating bar and content name clamped to a maximum width. The grid elements behave as text would, using the available width and passing to the next line if they do not fit.
- Usage of the TinyMCE plugin for editing of page type content (section 7.9).
- The DataTables plugin is used for navigation of groups, content and users (section 7.5). This allows for pagination, search and ordering of the list elements on multiple columns.
- SearchHelper, one JavaScript helper implemented for easing the creation of in-line search boxes (section 7.12). This was created with the specific purpose of selecting users from a list that pops up. It ended up as a reusable JavaScript object.

6.3 Request Controllers Layer

This chapter focuses on explaining the implementation of the base utilities that serve as a bridge between the sub-systems and the user interface. Functionalities of this layer serve as a basis for fundamental features for handling requests made to the server, like authentication, error pages, and asynchronous behaviour of the user interface.

6.3.1 Authentication and user session

User authentication is the process of having a user sign in into his/her account, entering the email and password as credentials. The user password is hashed, using the bcrypt encryption algorithm (Hashing function), on the server and checked if it matches the correct credentials. If the user gave the correct combination of credentials, he/she accomplished the authentication. If these credentials are not correct, the failed login is recorded in the “failedLogins” database collection to block excessive number of retries, avoiding brute-force attacks (trying to find out correct credentials).

The session of the authenticated user is implemented through a cryptographically secure token passed on Session and Cookie variables. The Session variables persist until the browser window is closed, while the Cookie variables are stored by the browser and persist for a designated time. Using these tokens, it is checked if they match with the ones stored in the “tokens” database collection.

To better secure the application it should make use of HTTPS to secure communications between the user browser and the application server and to prevent hijacking attacks like getting input from users on forms. The Play framework provides HTTPS functionality with a self-signed certificate (not verifiable by a trusted entity).

Each object on the platform has its own associated *AccessControl* object, which defines the roles that the user needs to have in order to view, open and do other operations with the relevant object. The relation between the authenticated user and his/her roles defines the authorization to take actions.

6.3.2 Synchronous and asynchronous requests

The framework enables easy checking for whether requests have been made through AJAX or not. In this way, the controller applies its logic for accomplishing related operations and checks if the request was asynchronous. For asynchronous requests, the controllers response is only part of the interface (usually to present on the main interface section), instead of the full page with all the interface sections.

A similar approach is used for submitting forms asynchronously. If forms are submitted as asynchronous requests to the server, the response will be a JSON object, while on synchronous form submission there is a redirect to an appropriate page. When doing a redirect, a *flash* variable is used for sending the result of the form submission to the page that will be presented. A *flash* variable is a variable passed through the HTTP header, like the session and cookies variables, but only is sent on the next request. Also, *flash* variables are a readily available functionality of the Play framework. Note that responses to forms have a text message and a label indicating status: success, info, warning, error, etc. On the asynchronous case, the message is passed on the sent JSON object and shown using the *toastr* plugin, showing one notification.

There is also a set of Java classes that are models for the forms used on the system, these use the tools provided by the framework for validation of user input. These verifications are such as checking required fields, emails validation and other values validations.

6.4 Components Logic Layer (Secondary Components)

During this section and the following one,section 6.5, the implementation of the components logic layer of the system is presented. First, in the current section, the secondary components implementation is explained and then how those are used by the primary components.

6.4.1 Example diagram

In this section, several diagrams are used to describe the implemented components or sub-systems. These diagrams show the related data models and some base operations that can be applied to that data, giving an overview of how each of these components work. A simple example of this is given in Figure 6.5.

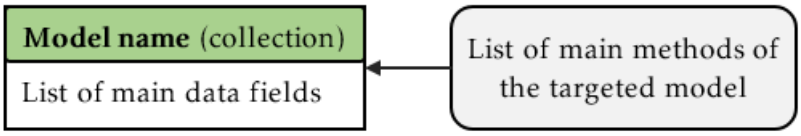


Figure 6.5: Example diagram.

6.4.2 Tags and metadata

The tags and metadata system implementation is based on **triple tags**, also known as *machine tags*. This is a notation for metadata with three parts: namespace (category), predicate and value. Thus, the ***metadata*** objects can be described with these three elements combined (**namespace : predicate = value**). As namespace and predicate identify a tag type, these aggregated values are used to identify a ***Tag*** (*TagId = value*). Tag and metadata instances are described in figure 6.6.

The value of metadata elements depends on the type related to this tag, identified with *MetadataType*, which can either be "ANY" or "COUNTER". Metadata objects have a current value object (MetadataObject) and also a list of values. In the case of the "ANY" type the list of values is used as an archive for previous versions of the value. On "COUNTER" type of metadata, the current value is the counter integer, while the values list holds the archived occurrences (i.e., on "feedback:vote_up" the list has the ObjectId of each user that voted up). All of these MetadataObject instances have a timestamp associated with it to keep track when these metadata values were created. Part of the **Metadata** objects are embedded inside its parent object, on a MetadataContainer, a map that stores current metadata values for fast access, while the complete **Metadata** is stored in its own database collection.

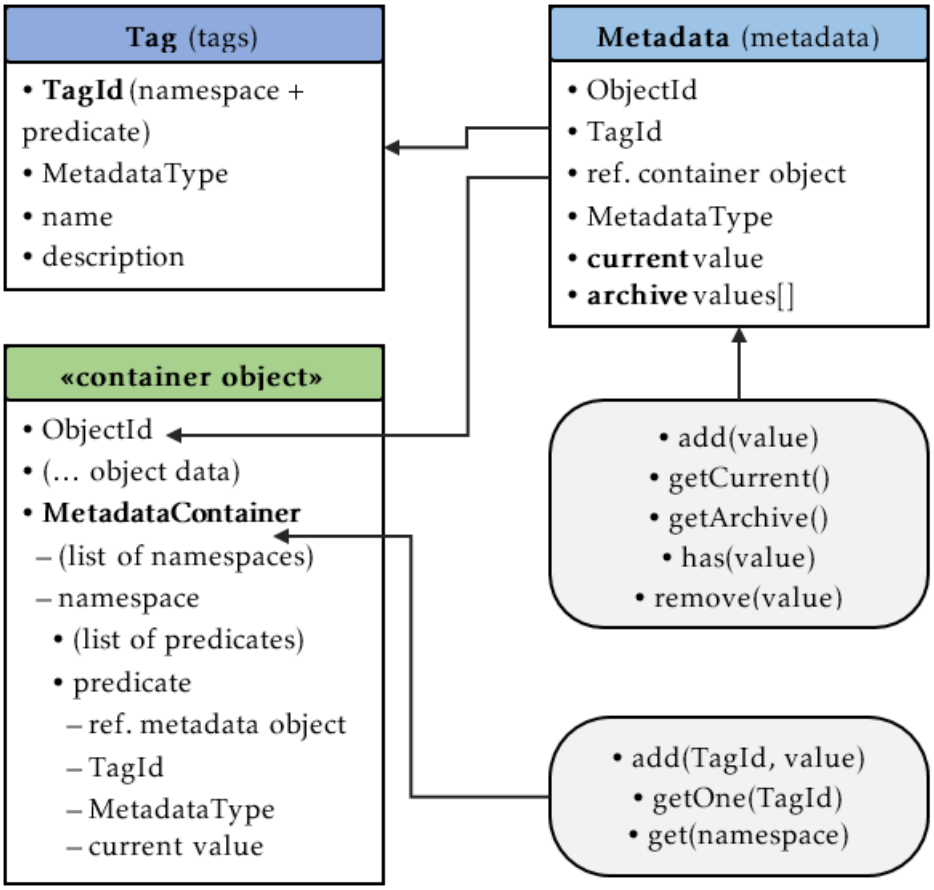


Figure 6.6: Tags system data and operations diagram.

6.4.3 Access Control

The implemented system for user access control resembles closely a strongly discretionary Role-Based Access Control (RBAC) model, as users are the ones controlling membership on groups and roles in them. These group roles are such as group administrator, regular member, group owner, etc. Some other roles are system-managed, automatically assigned roles like highlighted user with high rating, or untrusted user with suspicious behaviour (reported several times). The model for the data, always embedded on its parent object, is described in Figure 6.7.

Each *AccessControl* object defines the roles that the user needs to have in order to view, open and do other operations with its parent object. These roles have multiple domains, which are:

- *Global* level has platform user, public, guest;
- *User* level has platform-wide roles of a user;
- *Container* level identifies users within a context, like a user as member of a group or user is author of content;
- *Group member* level is a subset of *container* roles for user roles inside a group.

The lists of roles on *AccessControl* define the required roles for visibility (appearing

on the website search) and for access (permission to open the object), while other permissions are dependent on the parent object type. The permissions are of either one of these types: read, write or request. Note that there is a special *AccessControl* object, *AccessControlPlatform*, an hard-coded object that defines the permissions of users on the overall system (seen in Figure 6.7).

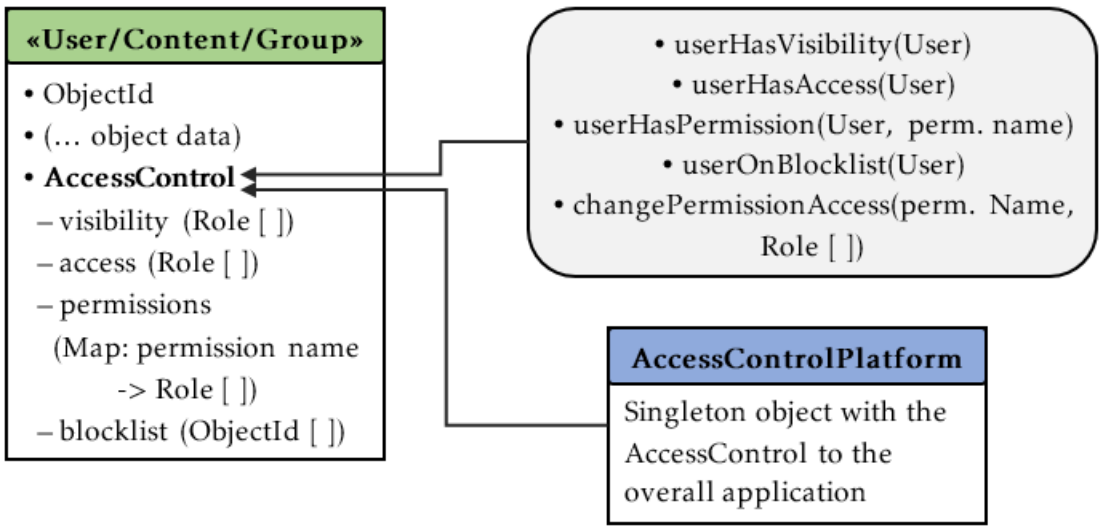


Figure 6.7: Access control data and operations diagram.

The implemented solution lacks the sessions described in the base RBAC model [53], therefore not applying the principle of least privilege, as the users are directly related to their roles and consequently their permissions. The association between the components (i.e., user has role X that gives permission Y to object Z) is not abstracted or secured cryptographically in any manner, which it should do in a commercial solution to avoid tampering of these relationships by someone with access to the system.

User permissions stack up when users receive new roles, as permissions are given to an array of roles. The only kind of constraint applicable in the system is a block list, as a very simple ACL, with a list of the users blocked from that object. Even so, the roles themselves do not have a defined hierarchy for inheriting permissions.

6.4.4 User actions and requests

In order to have a way to create requests on groups and to better visualize the permission system in action, mostly to track which actions have been taken. Actions taken are also stored in notification feeds, passed to these feeds through a model of actions Fanout. The actions fanout algorithm handles delivering the actions associated with each notification feed, this is further explained in the notification feeds section. The data model for actions is described in the Table 6.1 and actions have one of the following possible types (defined in enumerator *ActionType*): read, write, request, response. The type and verb of the action define what kind of action it is and what relates to the permissions on the access control

system. Some examples of these combinations might be: type = "RESPONSE", verb = "REPLY_YES"; type = "WRITE", verb = "VOTE_UP_CONTENT".

Of course some combinations of action type and verb make no sense and should never happen, but the creation of these actions is well defined using a *factory* (design pattern) class. The *ActionFactory* has all the valid methods for creating *Action* objects, checking validity of given parameters. Creating these actions, as most objects that are very complex or have a multitude of parameters for initialization, is accomplished using the factory pattern: separate class with functions that handle the creation of every kind of action and its validation. Responses are a special case of action, with their own action verbs: "REPLY_YES" and "REPLY_NO". A reference to the response action is stored in the respective request action, as well as the target of the response itself is always the request, so the two actions are linked to each other.

Table 6.1: Action model.

Data type	**Name**	**Description**
ObjectId	*id*	Identifies each object
ActionType	*type*	Type of action
ActionVerb	*verb*	Verb that identifies action taken
ObjectId, String	*actor*	User who made the action
ObjectId, ObjectType, String	*target*	*(optional)* Target of the action
ObjectId, ObjectType, String	*data*	*(optional)* Data object related to the action
ObjectId, String	*context*	*(optional)* Group where action takes place
Boolean	*response*	*(optional)* Response value to this request
ObjectId	*responseId*	*(optional)* ID of the response to this request

6.4.4.1 Restraining actions with permissions system

As stated previously in section 6.3.1, the access control system defines which actions the user has permission to do with each object or on the platform itself. There is also the possibility of actions to be requested, so that a user with further permissions decides to apply or not that action. With these features, it has become important to have a way of keeping track of requests and their responses status.

With the restrictions on actions being enforced by **AccessControl** and the user's corresponding **Role**s list, a relationship was created between **Permission**s and **Action**s using the *ActionType* and *ActionVerb* variables. This allows to check if the user can do or request a certain action, as illustrated in Figure 6.8.

The process of making requests and responding to them is shown in Figure 6.9. This was mostly created for users outside groups to be able to share content with the groups, yet the members of a group decide if they want it or not. Another goal was to have suggestions of changes inside the group that could only be decided by users with higher permission roles.

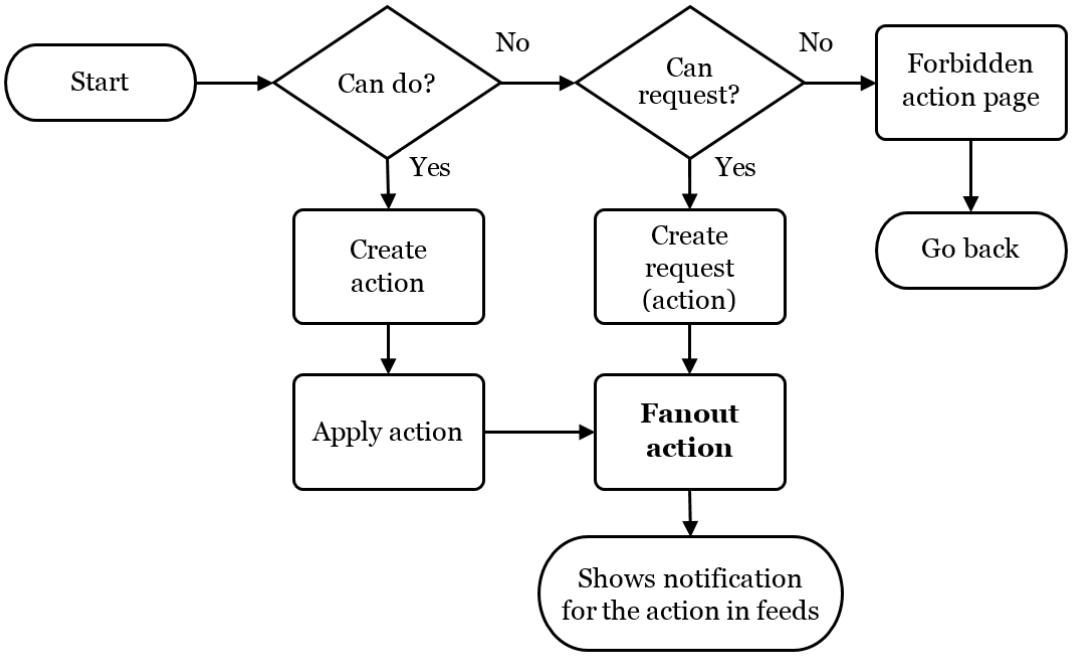


Figure 6.8: Trying to take an action, according to user permissions.

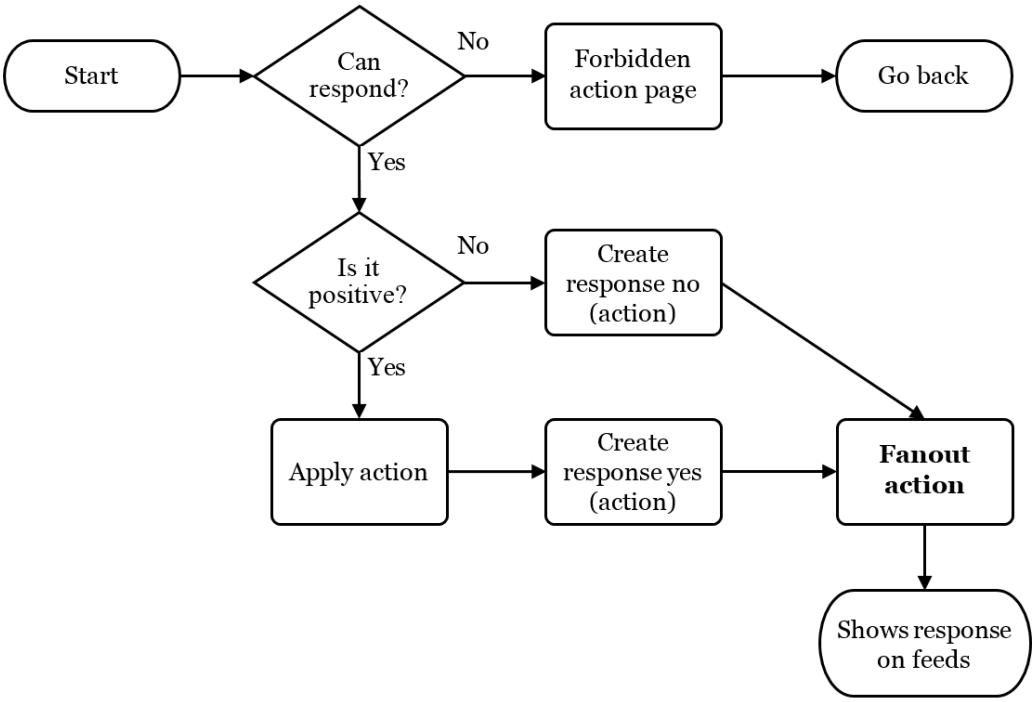


Figure 6.9: Responding to a request action.

6.4.5 Notifications feeds

After an **Action** has been taken, requested or responded to, it is Fanout into the notification feeds, which means it is delivered to the notification feeds that follow the target object of the action, as shown in Figure 6.10. Users by default have a notifications main feed shown on the *Home* page with all the actions related to the user, its contents and its groups. In order for the user to be timely notified of important actions, another notifications feed appears on the sidebar with a counter of unread important notifications.

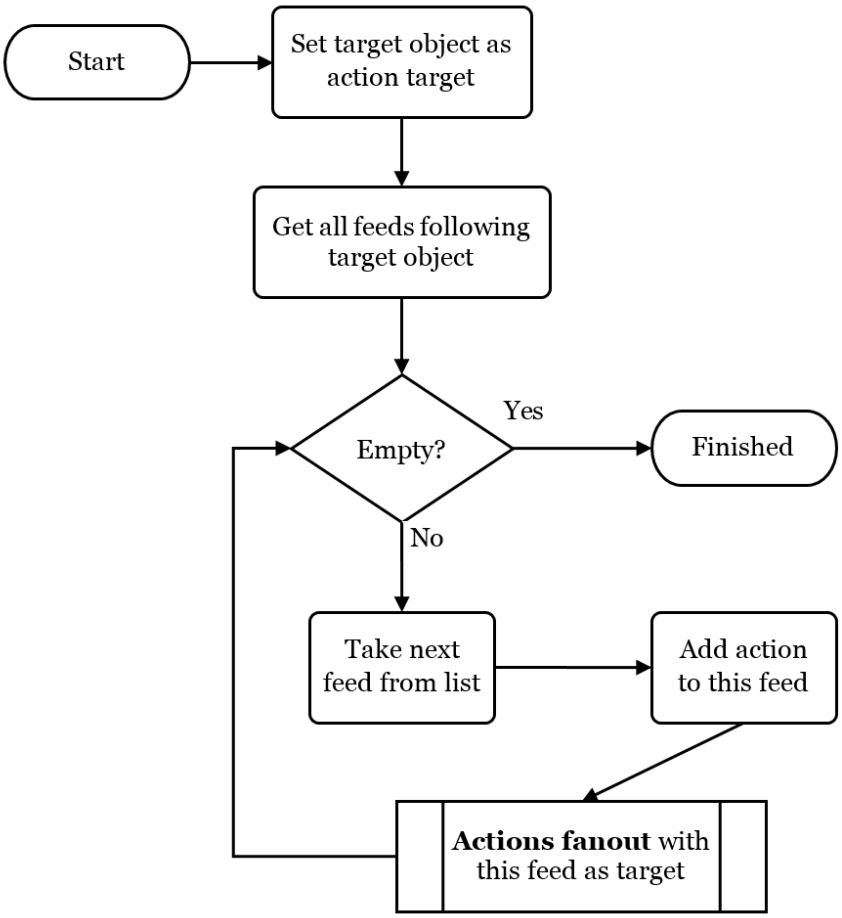


Figure 6.10: Actions fanout (placing on feeds).

The main component of the notifications feed (Figure 6.11) is the list of **FollowingObject**s, which has the reference to a platform object. All the actions targeted to an object followed by the feed, except those with **ActionVerb** that is on the filter list, will become a notification on the feed. Note that feeds can also follow other feeds. Such is the case with the important notifications sidebar feed, following the main feed but filtering less relevant actions.

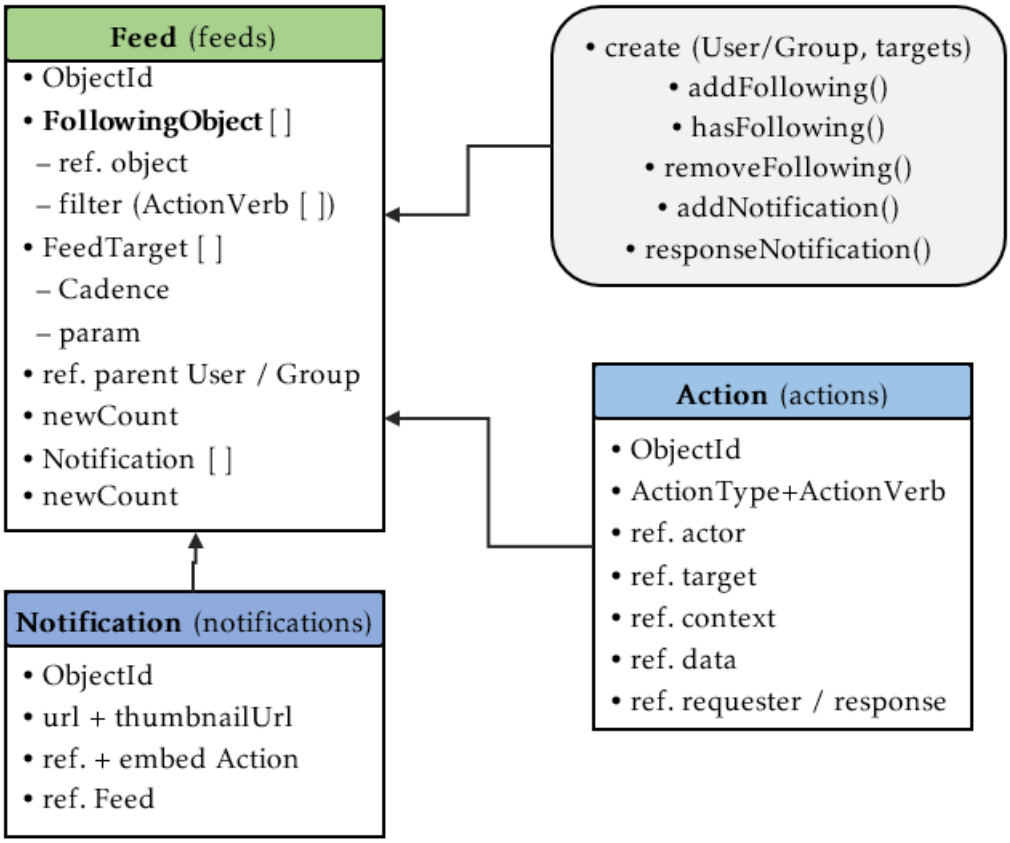


Figure 6.11: Notification feeds data and operations diagram.

6.4.6 Rating

As already mentioned throughout the chapter 4, assessing the quality and relevance of the content is very important for users searching for meaningful information. To evaluate the quality of content, a bar was created next to each content link to indicate the rating of this item; full bar means it is top quality content, while empty bar means it has only negative feedback.

The feedback elements and rating values are stored using the metadata system. When feedback is given to a content, its corresponding weight value is added to a counter (see section 4.5.2), with this counter of feedback values received other factors like content age are taken into account to get the percentage rating value. This rating counter is stored on the content metadata with tag id "rating:raw".

The percentage value is also calculated at that moment, using the combination arc-tangent function and linear interpolation already described as the *"rAggr4Avg"* method in the rating Chapter (4). The percentage rating result is stored on metadata with the tag id "rating:percentage". The same process is done after this for computation of the rating values of the user creator of that content element.

6.4.7 Messaging / Chat

The chat system was implemented as a simple collection of text messages from a user and targeted to another user or to a group (for the group chat). For presentation of the messages between users, the system does a query using the aggregation pipeline of MongoDB to filter all the messages where the current user is involved (sender or receiver), grouping all those messages for each unique user the current user has messages with. The user interface is implemented as a sidebar panel that automatically refreshes content periodically.

6.4.8 Administration

The system offers one page for users with administrative rights on the platform, where these can manage several aspects of the platform. This is used to enable/disable obligatory administrator verification when new groups are created, make the new groups verification, enable/disable user signup and also for inviting users to the platform.

6.5 Components Logic Layer (Main Components)

This section shows the implementation of the main application components, the users, groups and contents, which are part of the components logic layer of the system.

6.5.1 Users

The data from users is stored on the "users" database collection and related data structures, shown on figure 6.12. As already mentioned, users hold their email and password hash as credentials for authentication on the application. They are also identified by a unique user name, can upload a picture and their email must be confirmed.

Users have their list of roles and AccessControl object, explained with further detail on the access control section (6.4.3). Most systems and data objects link to users, these do not have much other data besides authentication credentials. Other data that could be added would be other personal information and settings regarding the user.

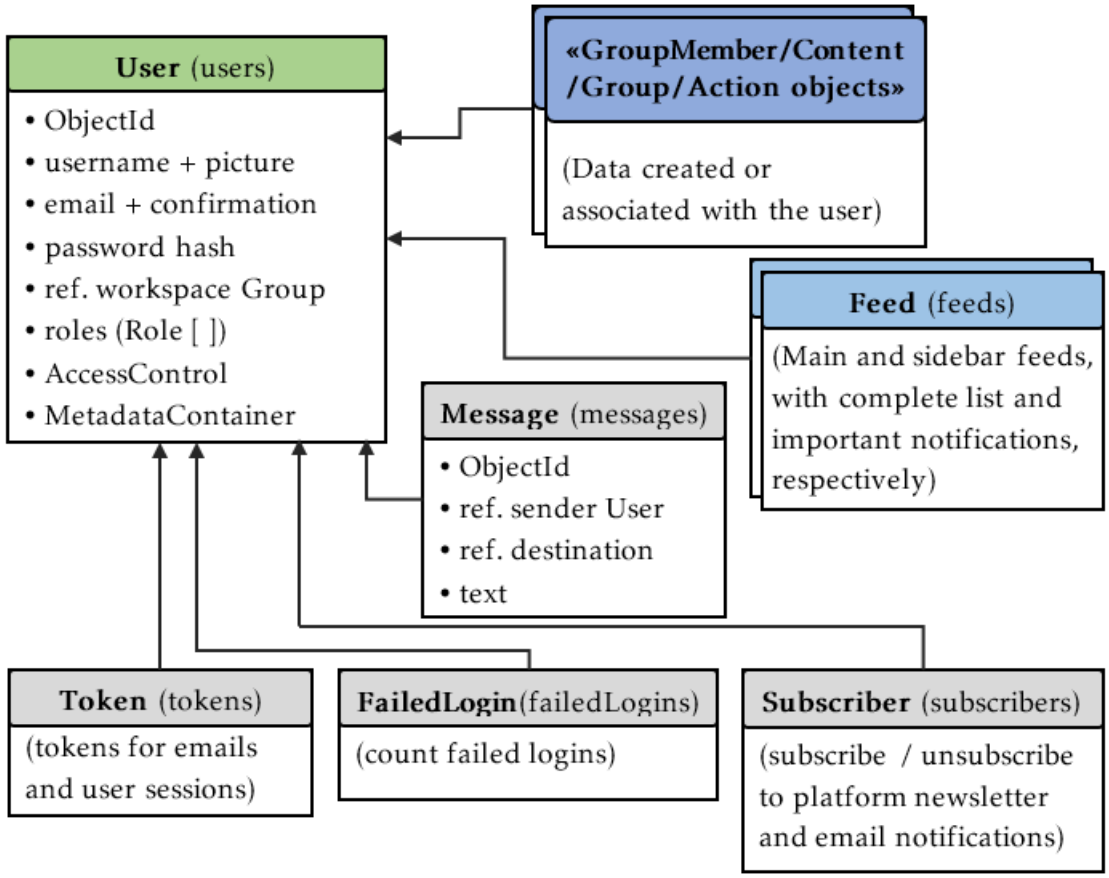


Figure 6.12: Users data diagram.

6.5.2 Groups

The groups are the main organizational structured of the system, containing the content of the platform and placing users together as group members. Both the content and members on groups are stored in their own data structure, ContentSection and GroupMember, respectively. The structure for the data and base operations on groups is presented on figure 6.13.

In the same way as with the content, there are two pre-sets for the AccessControl object: public and private. In order to have control of the groups created on the system, there is an option for requiring administrator users of the platform to verify new groups. In order to describe the purpose of each group there is a text description for it, ideally there would be more information fields like language, tags and related groups, but for simplicity the description serves that purpose.

6.5.2.1 Members management

The group members have several roles specific to their group memberships. For the purposes of the access control system and user interface there is only distinction between administrator roles and non-administrator roles. Without a user interface to add and edit

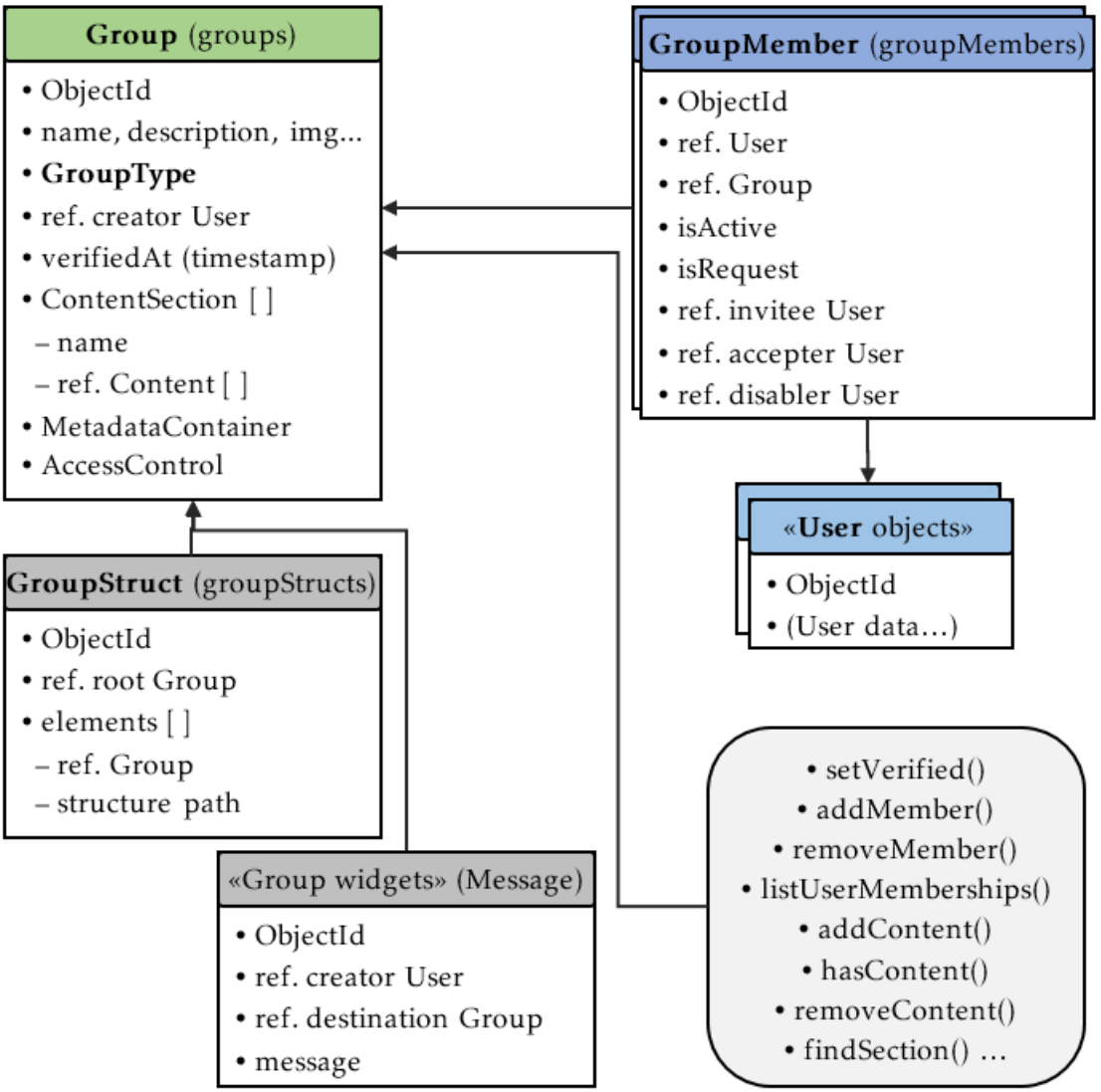


Figure 6.13: Groups data and operations diagram.

roles for groups, their permissions and clear rules for assignment to members, it would not make sense to hard-code the distinctions between similar roles. So, the group member domain roles "ADMIN" and "OWNER" give administrative rights and all the other do not. Only users with administrative rights can give or remove the administrator roles.

Users can request access to any group they have access to. Membership requests are visible to group members and those with the proper permission can respond to the requests. Note that these are the only requests that do not use the notification feeds as they were implemented before that feature. The data of requests is summarized on the GroupMember object of figure 6.13. Memberships can also be invited from group members and work in a similar way to the requests, but the invitation goes to the invited user main notification feed. After being a member a administrator can block a member (deactivate) and remove members from the group. Users can also leave groups, unless

they are the last member, which deletes the group itself.

6.5.3 Content

The content of the platform holds the data created and exchanged by the users and its structure is presented in Figure 6.14. There are only 2 types of content implemented, simple pages and files. Pages are rich text created on an WYSIWYG editor, stored and presented as HTML. Files are any type of file data, where viewers can be embedded on the web page to view the file contents without the need of downloading (this is only used to show images).

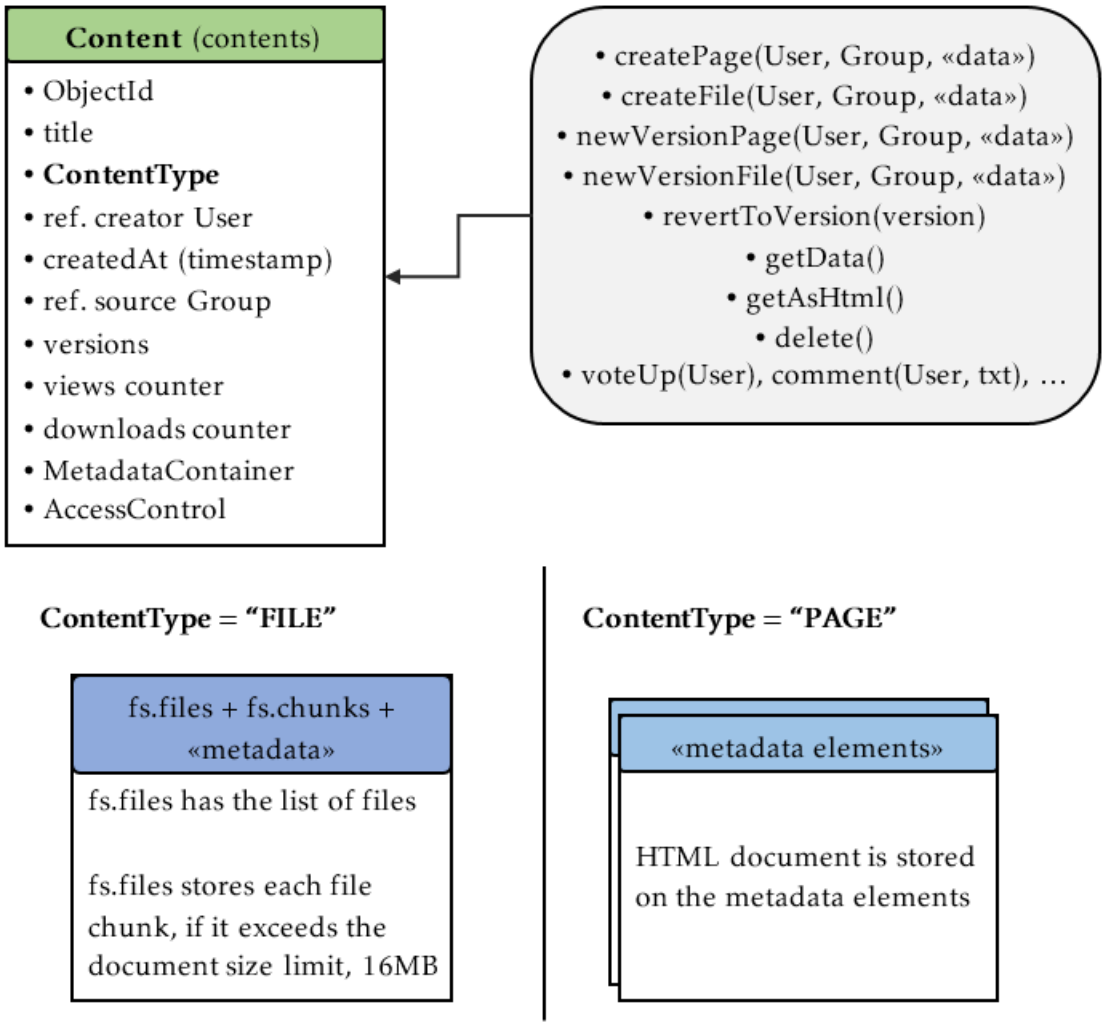


Figure 6.14: Content data and operations diagram.

The content files are stored using the GridFS file system functionality of MongoDB, and the references to the files data is stored on the content metadata identified by several tag ids inside the namespace "content". Page content is also stored on the metadata. This is done to enable having multiple versions of the same content, so that users can edit the content.

Even only implementing these types of content it would be a matter of front-end work and few logic changes to implement others, as the data of the content is stored using the tags system, it could for example interface with external APIs like file storage and sharing services (i.e.., Google Drive, Dropbox, etc.) or even create more functionality rich types of pages as a wiki structured page.

On each content element there is a sharing menu, it enables the users to easily share on social networks and email. It also allows to share the content with another groups on the platform, showing a list of groups. When a content is shared and the group receiving it accepts it, the AccessControl of the content is changed to allow access to members of that group.

Chapter 7

Results

This chapter presents the implemented system and its usage scenarios, discussing the resulting features. First the features related to users are presented, then relating groups, then contents, and finally other elements. The system was named **Ant Academy**.

7.1 Opening the web application

First, let us define the base web address as `"http://ant.academy"`, omitting this part of the address on the following URLs. When opening the base address (root address - `"/"`), the landing page is shown (Figure 7.1. On the landing page, clicking on the title or "Academy" button leads the user into the home page of the application (`"/app"` – Figure 7.2). Here users can login, signup or access sections that do not require authentication. This route (`"/app"`) also holds the main user notification feed, when authenticated.

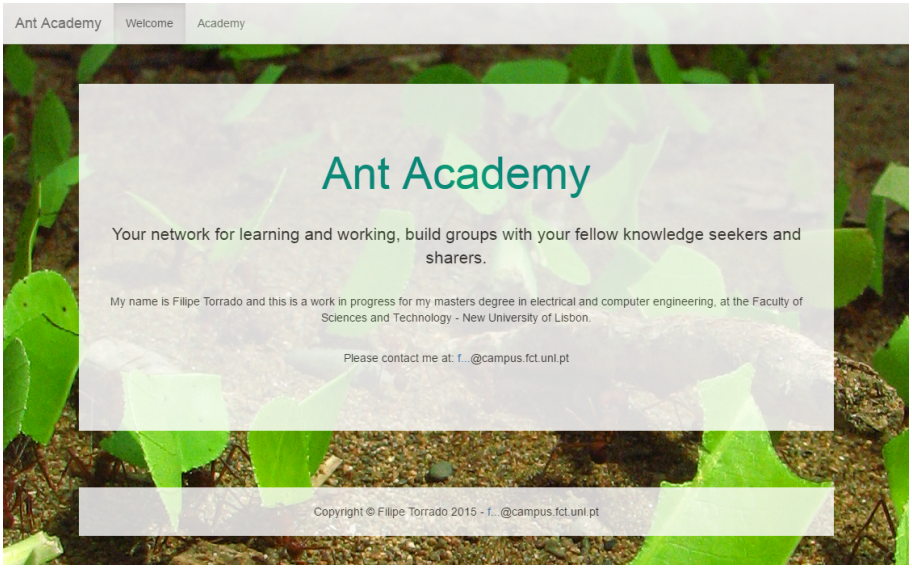


Figure 7.1: Landing page.

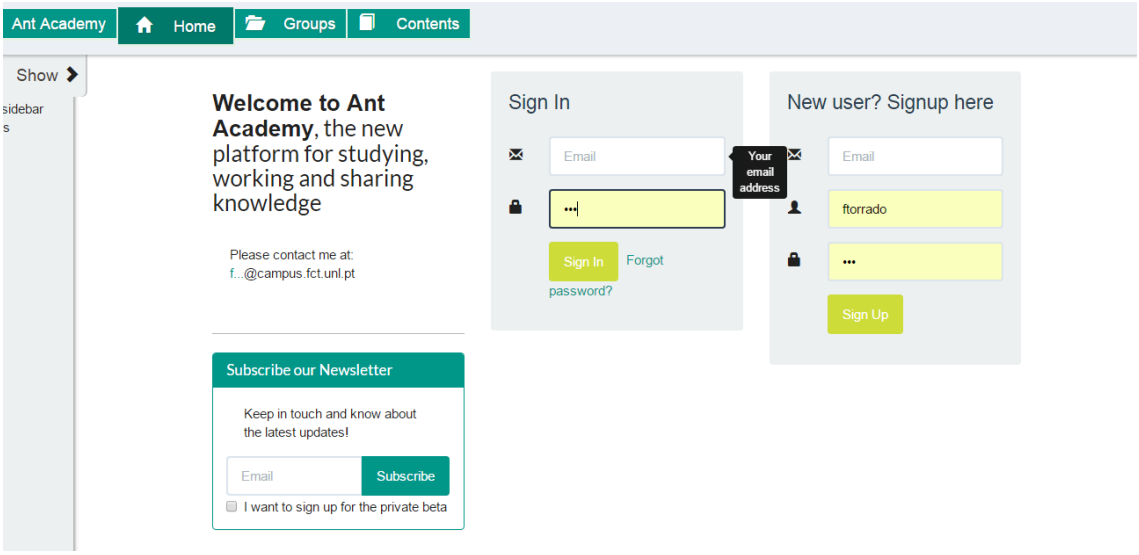


Figure 7.2: Login / signup page, home page for unauthenticated users.

7.2 User signup

There are two ways to signup to the system: using the form on the home page and being invited. After filling in the form on the home page with a valid unique email and a password, users will be shown a notice page (Figure 7.3), warning that an email was sent asking for confirmation of the provided address (Figure 7.4). After the confirmation is made, users are shown a success page (Figure 7.5) and an email confirmation with links to important pages so he/she can get started using the platform (Figure 7.6).

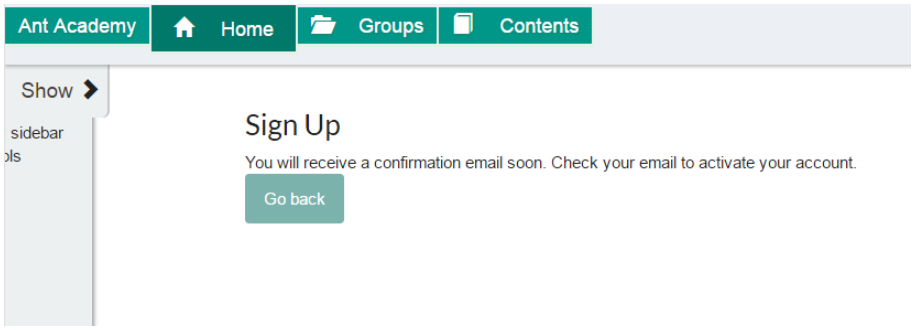


Figure 7.3: Note for email validation.

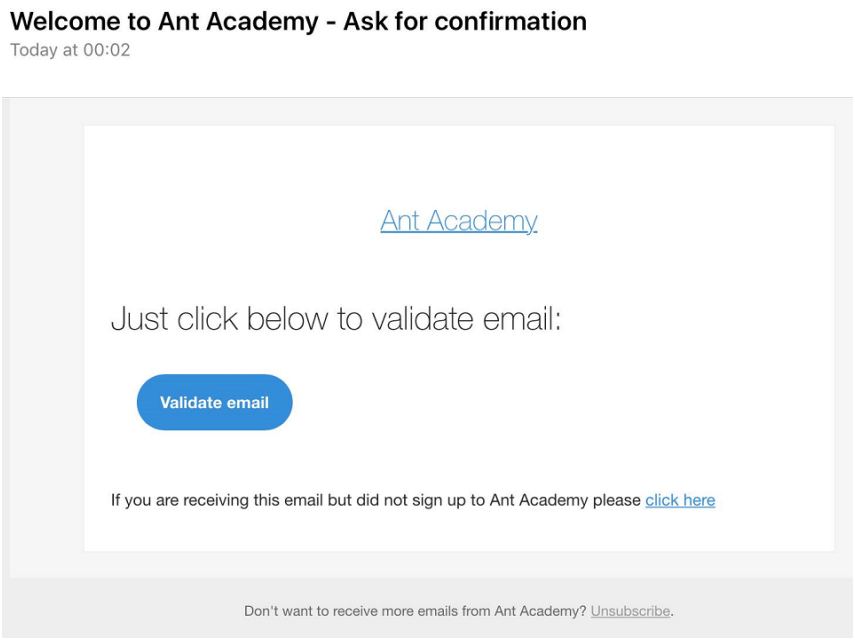


Figure 7.4: Validation of provided email address.

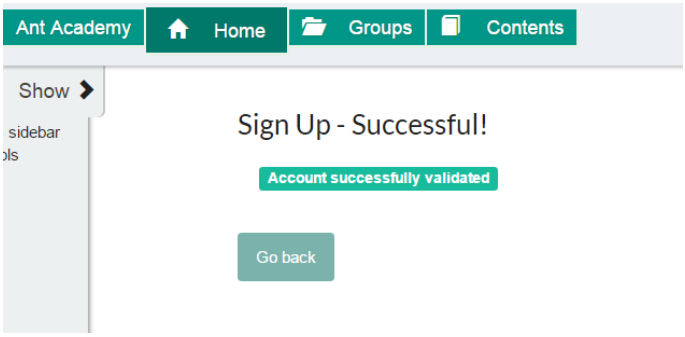


Figure 7.5: Confirmation that signup was successful.

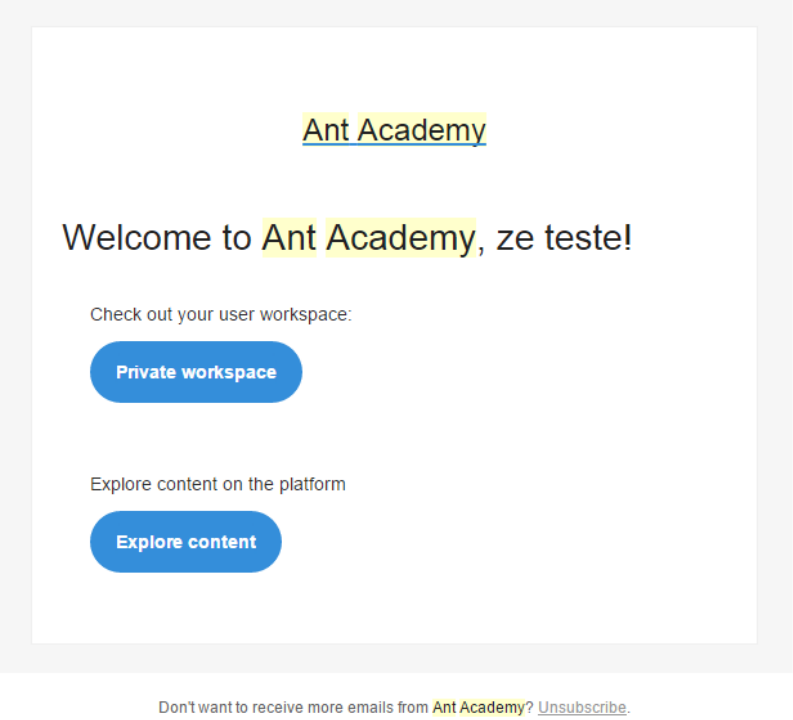


Figure 7.6: Welcome email after users sign up.

7.2.1 Invited users

Users can also be invited to the platform, receiving an email with the invitation. See section 7.11, with a form on the administration page for inviting users (by email address). After the invited user opens the unique link on the invitation email, it redirects to a page for selecting the user credentials and finish signing up, or reject the invitation. This counts as email validation, as this also uses a similar unique token scheme. Also, the same confirmation pages and email are shown after signup.

7.2.2 User settings

After the user has signed up to the system, he/she can change the user password and email on the user settings page.

7.3 Main user notifications feed

The home page, for authenticated users, shows the user's main notification feed. This feed has all the relevant actions related to the user, including requests. Figure 7.7 shows the feed for a user that just signed up. Figure 7.8 shows the feed for a user already with several actions and requests.

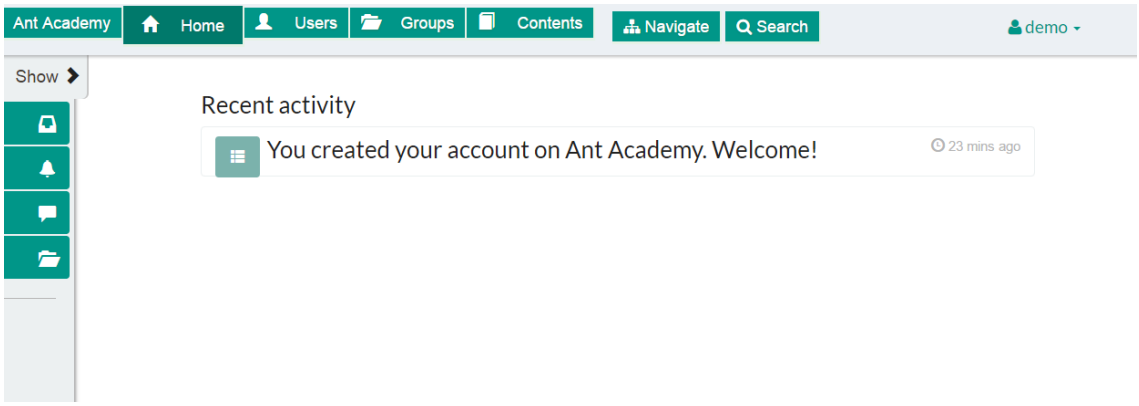


Figure 7.7: Home page after signup and login.

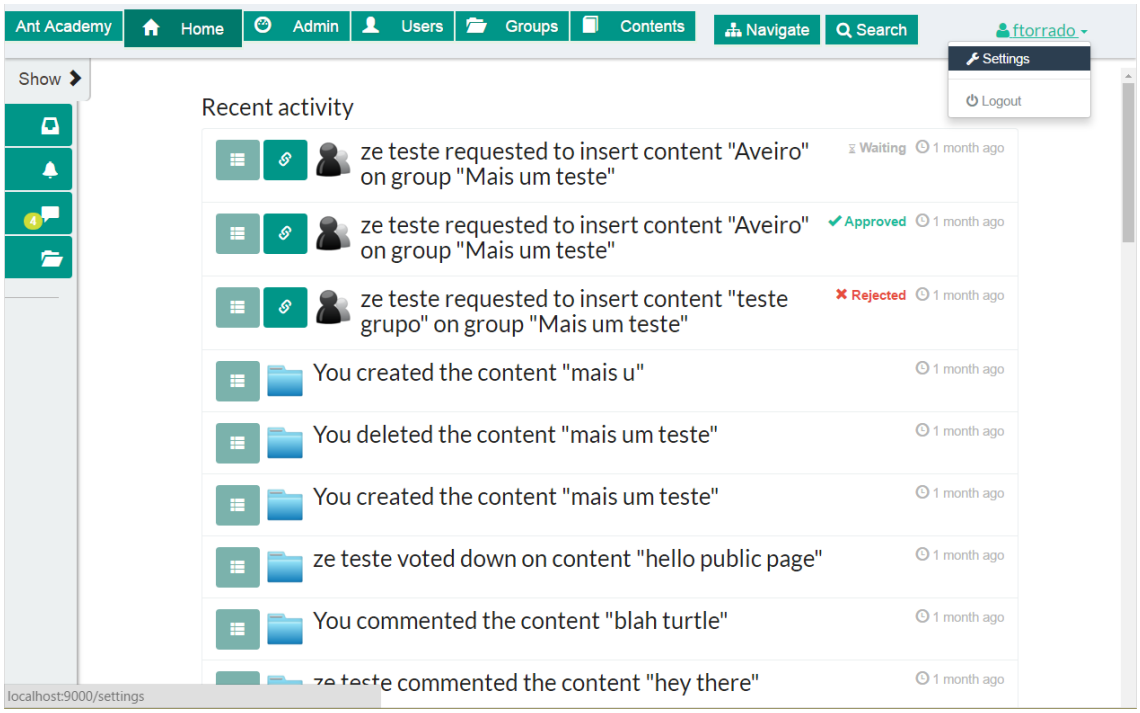


Figure 7.8: Home feed of user already with several actions.

7.4 Users section

Clicking on the header button to access the users section, a page is shown with the list of users that are colleagues on groups and several links related to the user profile (Figure 7.9). The user profile page, that can be seen by all users, is shown on Figure 7.10 and the page for editing profile is shown on Figure 7.11.

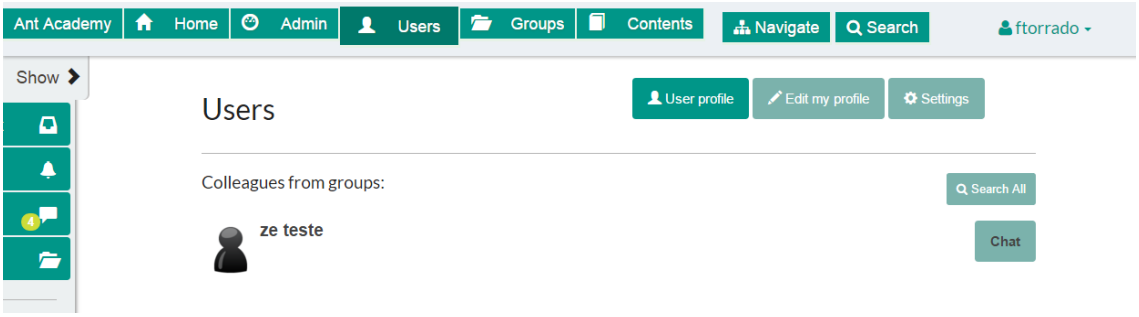


Figure 7.9: Main page for the users section.

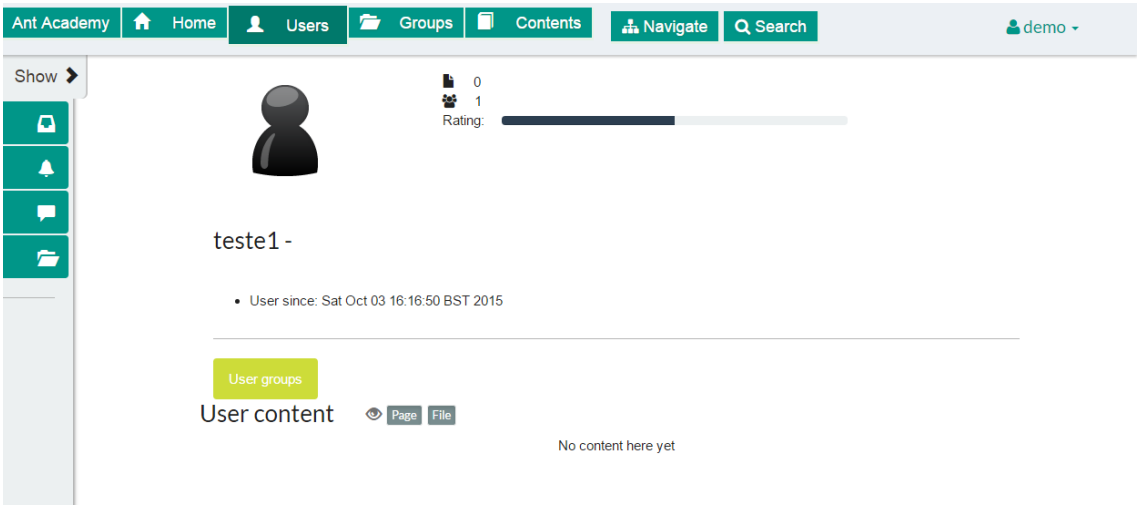


Figure 7.10: User profile page.

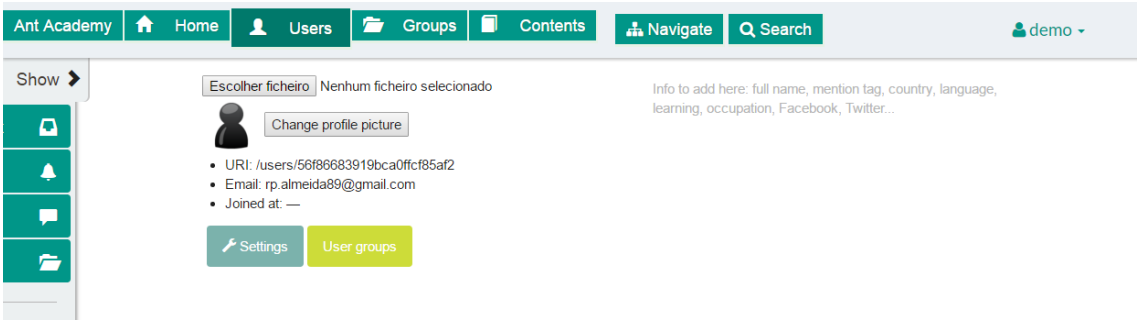


Figure 7.11: Editing the user profile.

7.5 Navigation and search pages

For easier browsing of user, group and content elements on the system, the DataTables plugin and Bootstrap tabs are used on the "Navigate" and "Search" pages. These pages are used to either **navigate** the elements related to an user (group membership and created content), or **search** all elements that are visible to it (Figures 7.12 and 7.13, respectively).

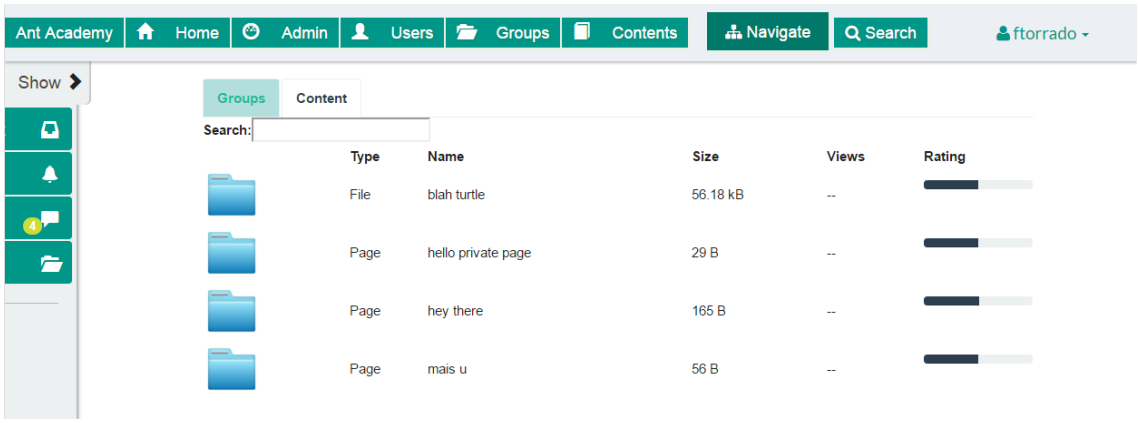


Figure 7.12: Navigation of content elements.

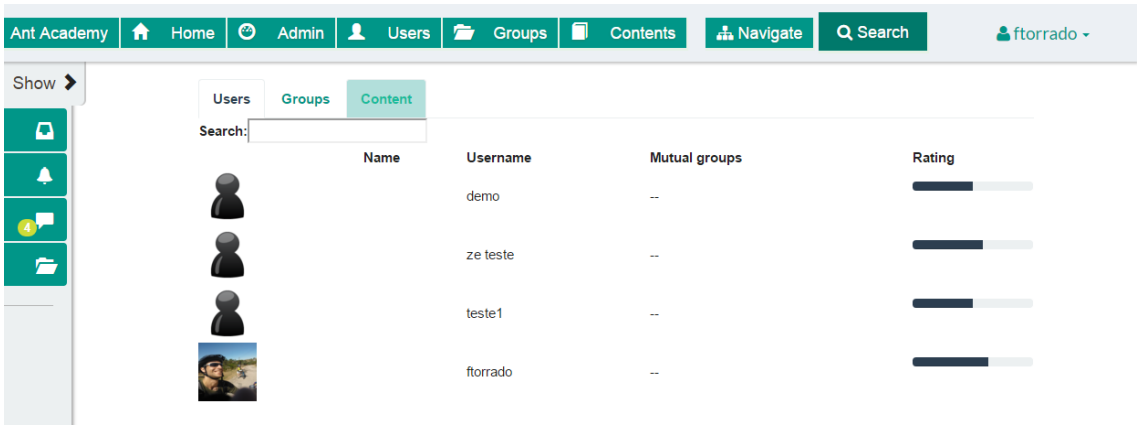


Figure 7.13: Editing the user profile.

7.6 Groups section

Going to the groups section, a list of the groups that the user has a membership on is shown, as well as other operations related with the groups (Figure 7.14). This enables users to easily access their private workspace group and to create new groups (Figure 7.15).

After creating the new group, it is empty and the user is the owner member, as shown in Figure 7.16. When the user tries to open a group that he/she has no access to, the page on Figure 7.17 is shown (same as content or other pages with no access).

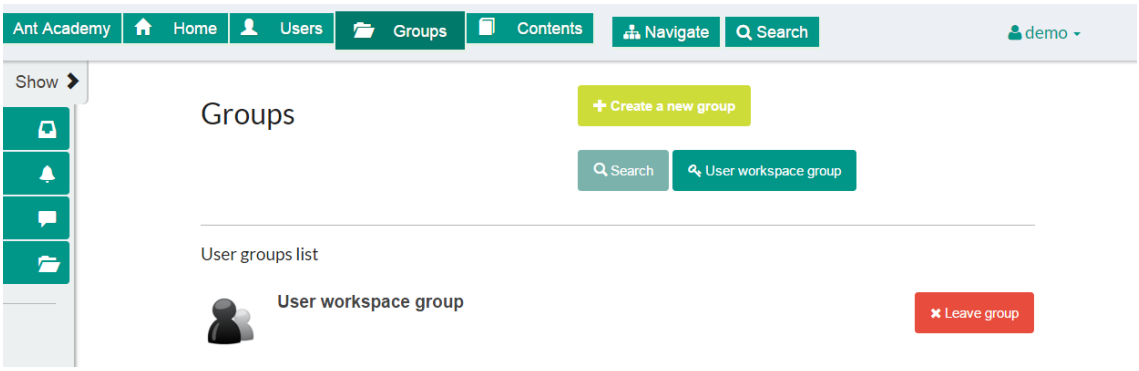


Figure 7.14: Groups section main page with list of user's own group memberships.

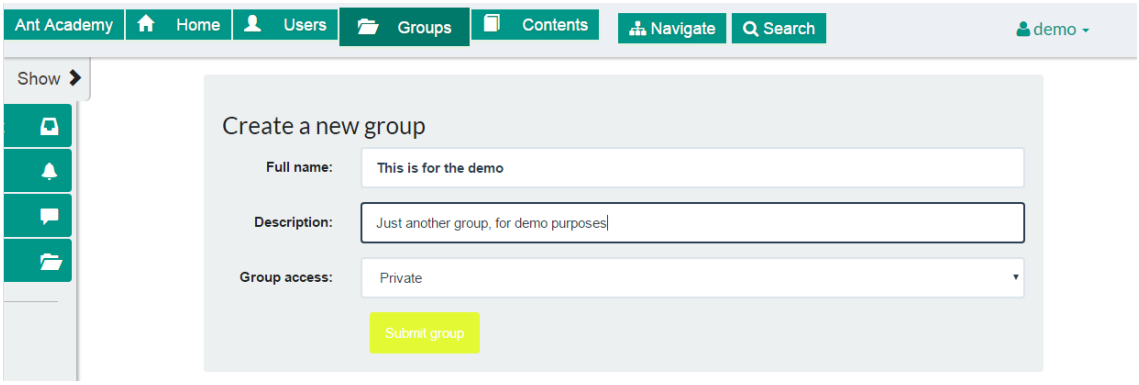


Figure 7.15: Creating a new group.

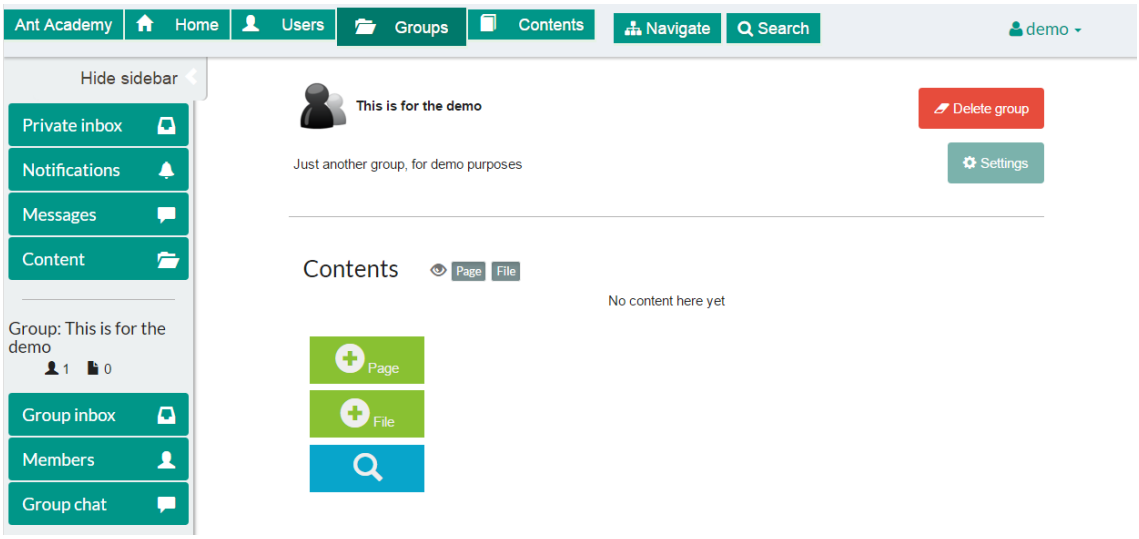


Figure 7.16: Group page after creating it.

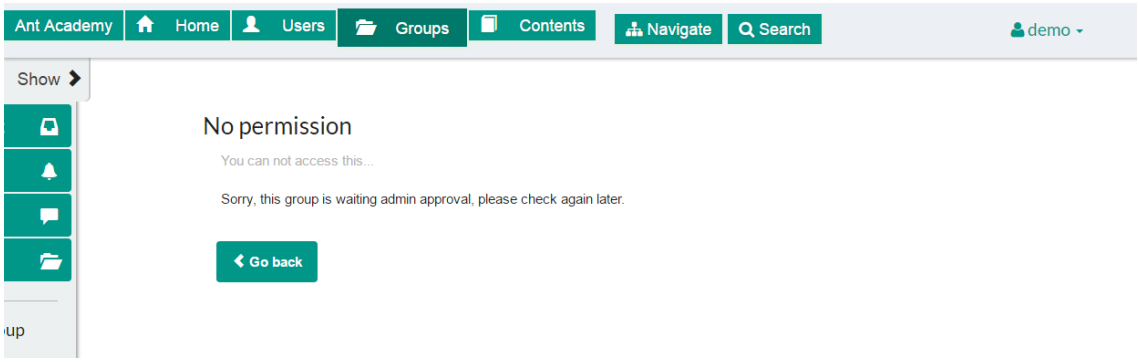


Figure 7.17: Group that the user has no access to.

7.7 Group memberships

7.7.1 Membership request

When users click the "request membership" button (Figures 7.18 and 7.19), a request is sent to the group for membership. Group members then decide to give or not membership to who requested it. This request is shown on the list of user's groups (Figure 7.20) and to the group members with permission, for responding to the request (Figure 7.21). After accepting the membership request, an asynchronous confirmation is shown and the user moves from the requests list to the members list, as seen on Figure 7.22.

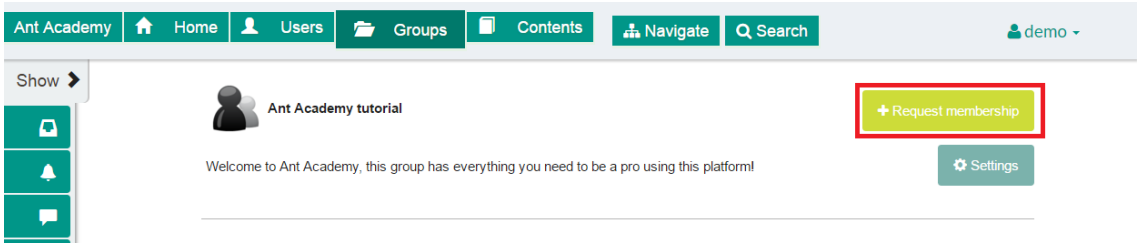


Figure 7.18: Group membership request button.

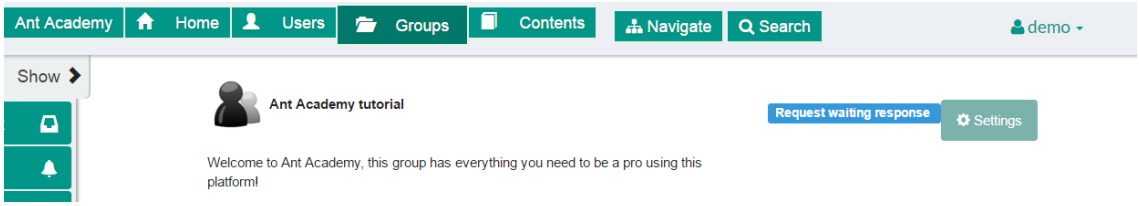


Figure 7.19: After pressing the request button.

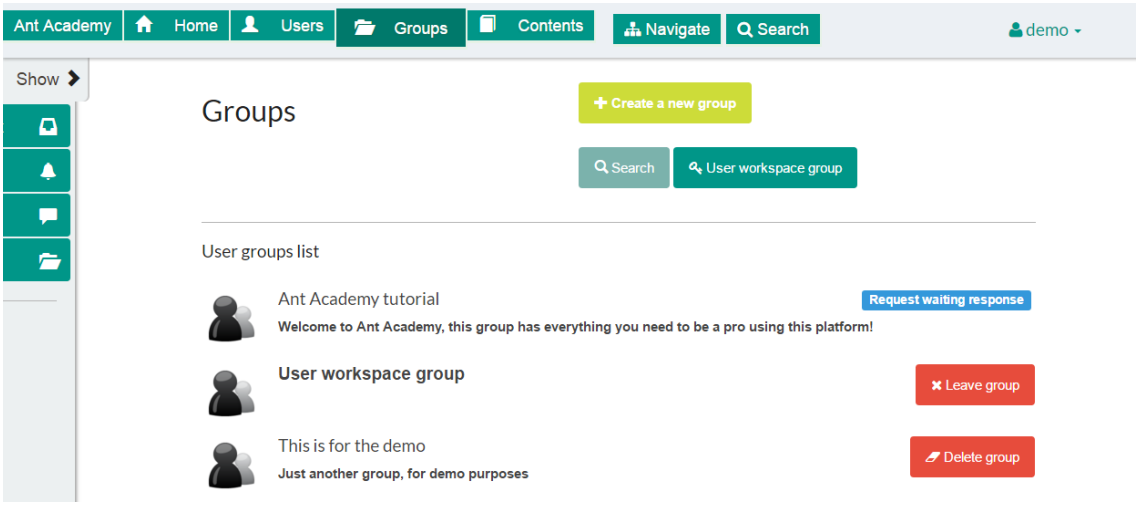


Figure 7.20: List of user groups with the created group and sent request.

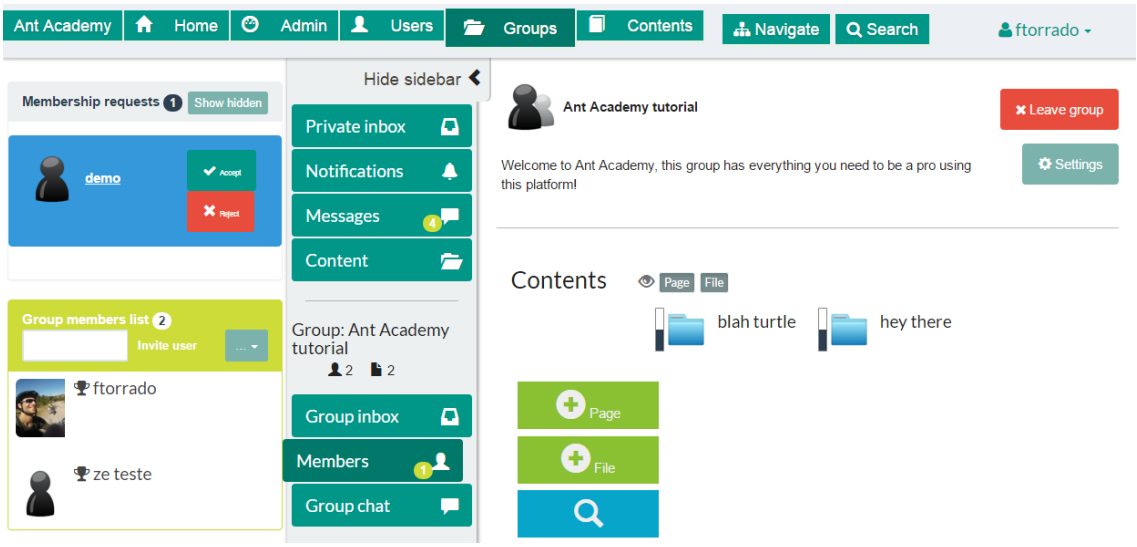


Figure 7.21: List of group members and membership requests.

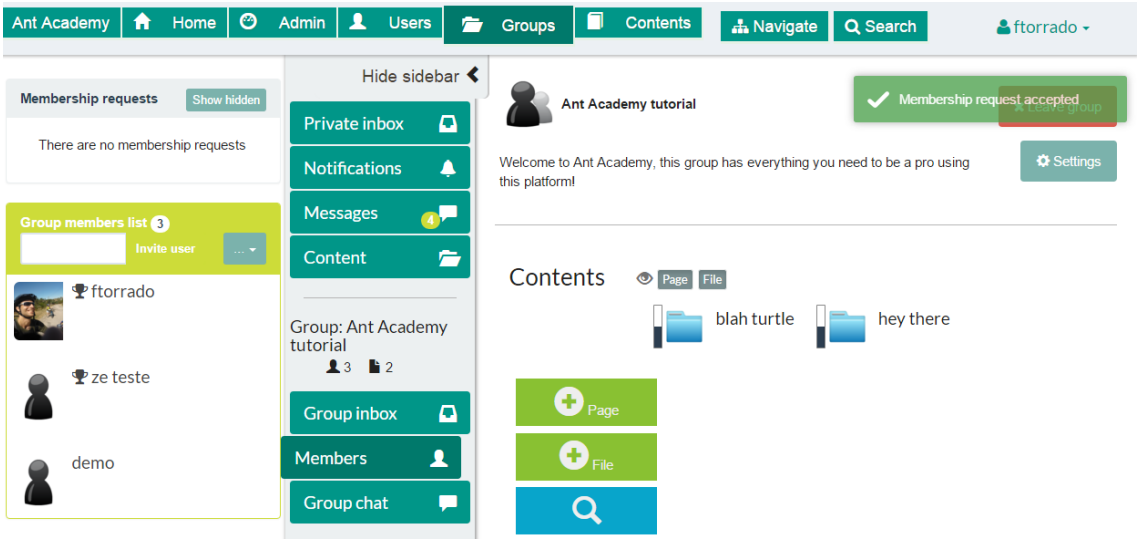


Figure 7.22: Right after accepting the user membership request.

7.7.2 Membership invitation

Membership invitations are sent within the members management interface on the group sidebar. Users are selected from the list shown with the SearchHelper tool and then the "Invite user" button is pressed (Figure 7.23). After the user is invited, he/she receives a request on the notification feed, for responding to the request.

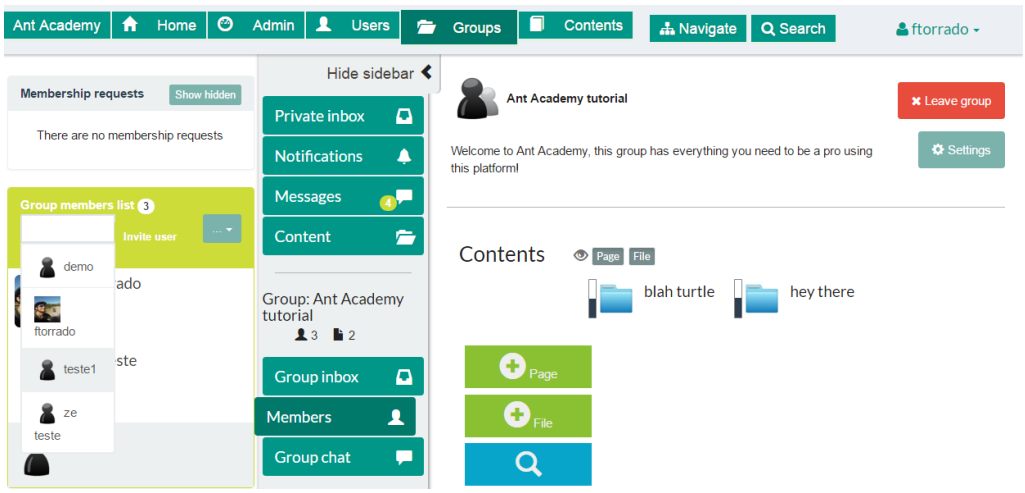


Figure 7.23: SearchHelper interface for selecting users to invite into the group.

7.7.3 Managing members

The users on the members list are selectable, the background of the selected element becomes greyed (might be too subtle to show on printed version). Several member management actions can be done, like promoting or demoting administrator role, with proper user permission. The Figure 7.24 shows the "demo" user selected and related operations.

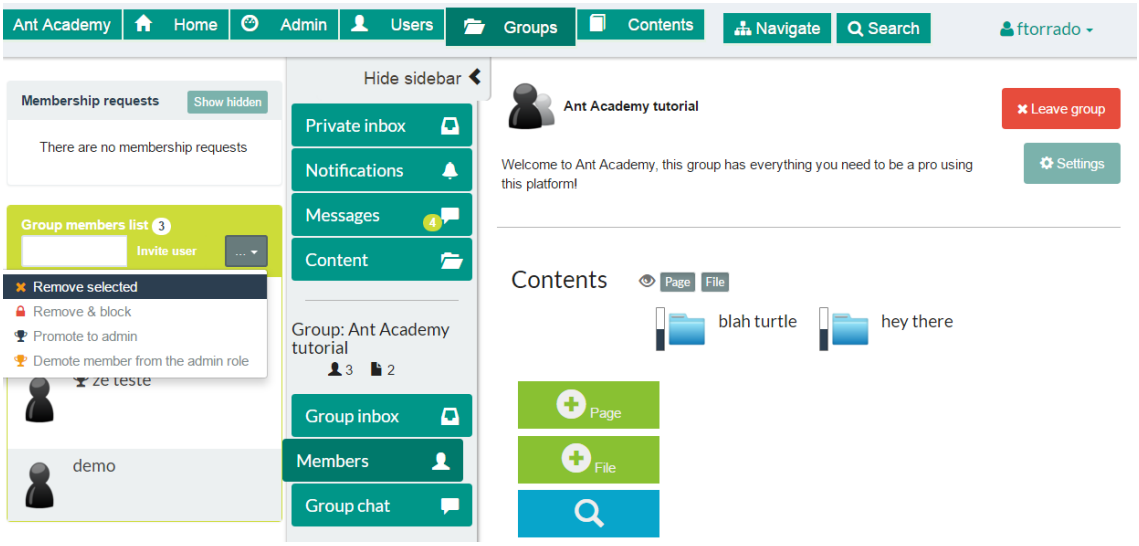


Figure 7.24: Management of group members.

7.8 Group management

For management of groups, a settings page was created (Figure 7.25) with form for editing information, list of members for page with member details and links to other settings. The groups can only be deleted when the last member leaves the group, so there is no control for doing it directly.

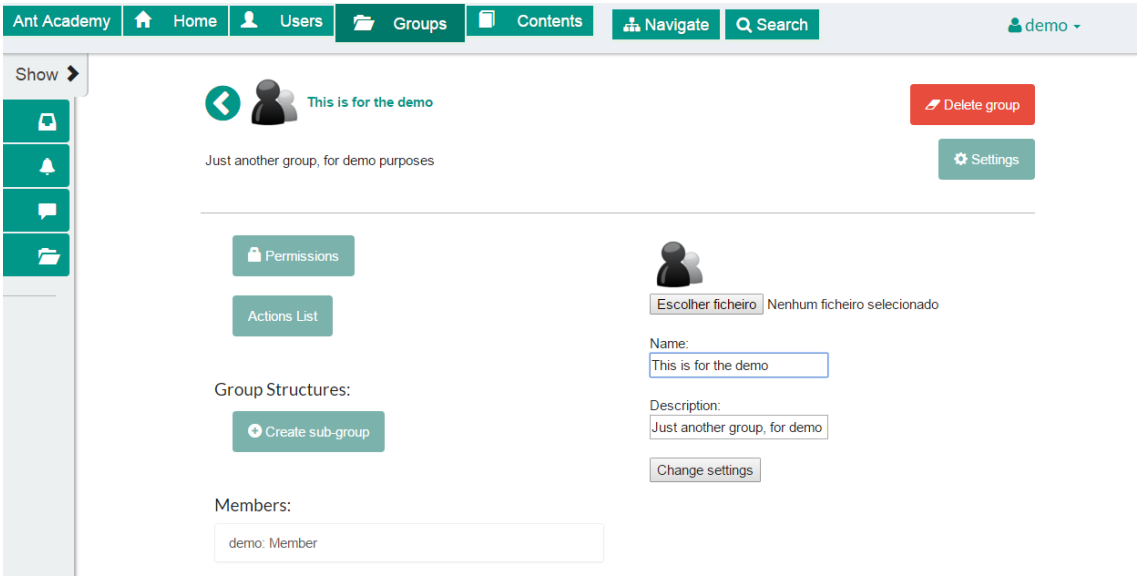


Figure 7.25: Page with settings of the group.

The main notification feed related with the group is shown on Figure 7.26. Creating sub-groups using the group structures was not implemented and the form for editing group permissions is unfinished (Figure 7.27).

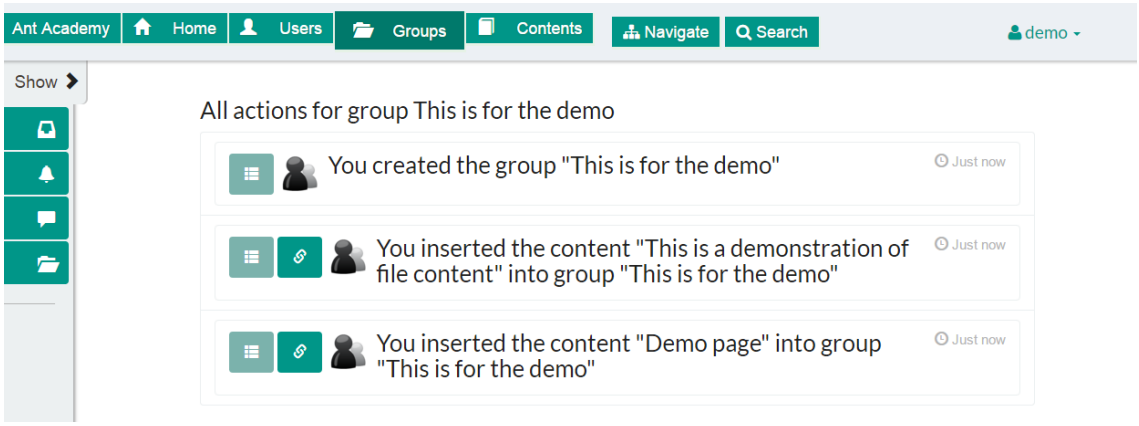


Figure 7.26: Feed with actions taken on the group.

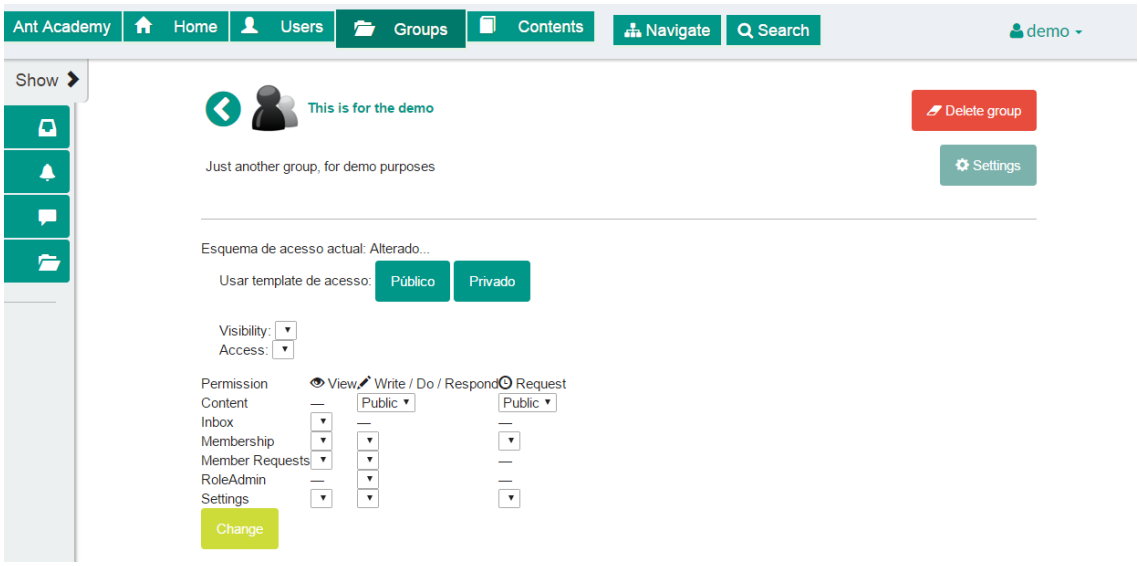


Figure 7.27: Unfinished form for setting group permissions.

7.9 Content section

The index page on the content section, shows a summary of content created by the current user (Figure 7.28). Users can create content either using the buttons on the top of the content section or using the buttons on the groups, creating the content inside the group. Figures 7.30 and 7.29 show the creation of content inside a group with types file and page, respectively.

After the content is created, users with access to it can open it. On the content page users can: vote up, vote down, comment, download, share and see list of groups where it is shared (requesting edit and reporting are not fully implemented). Figures 7.31 and 7.32 show the created file and page contents, respectively.

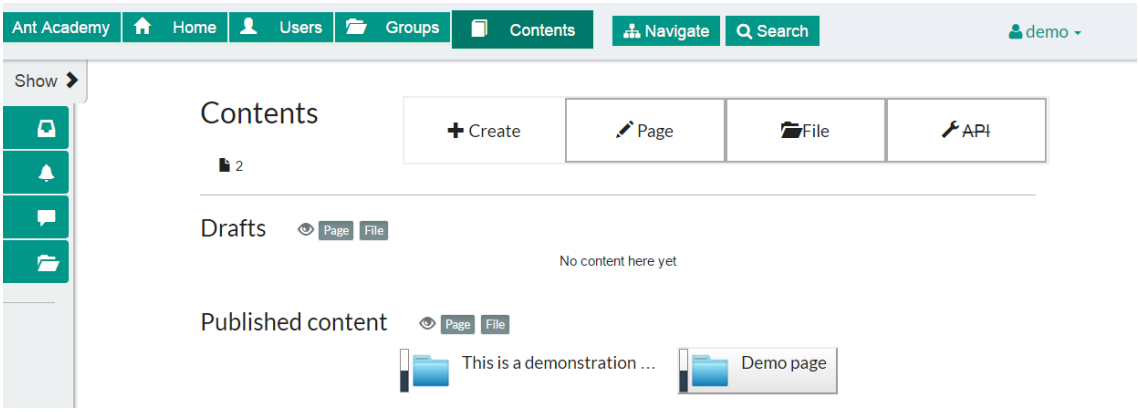


Figure 7.28: Main page for content section.

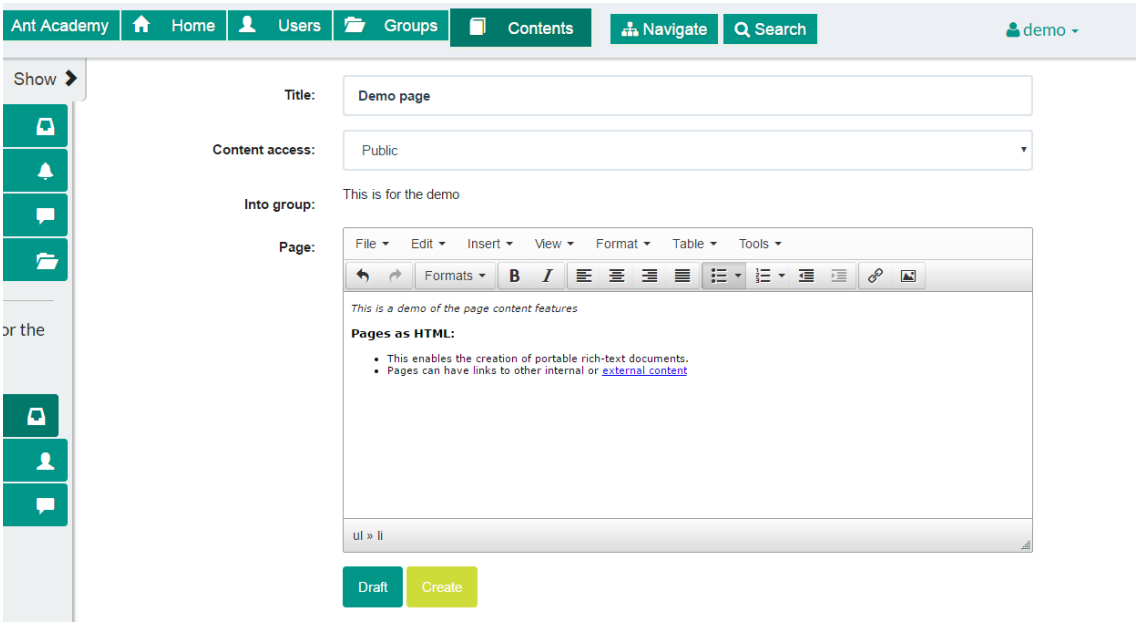


Figure 7.29: Creating a page content inside a group.

7.10 Sharing content between groups

For sharing the content to other groups, users make use of the share menu on the content, shown on Figure 7.33. This allows for easy sharing of the content into social networks and other mediums like email, as well as into other groups on the platform that are visible or accessible and allow requesting or doing sharing of content. If the user does not have permission to share the content and only to request it, this request is sent to the group notification feed and to the feeds of its members (that can respond to the request).

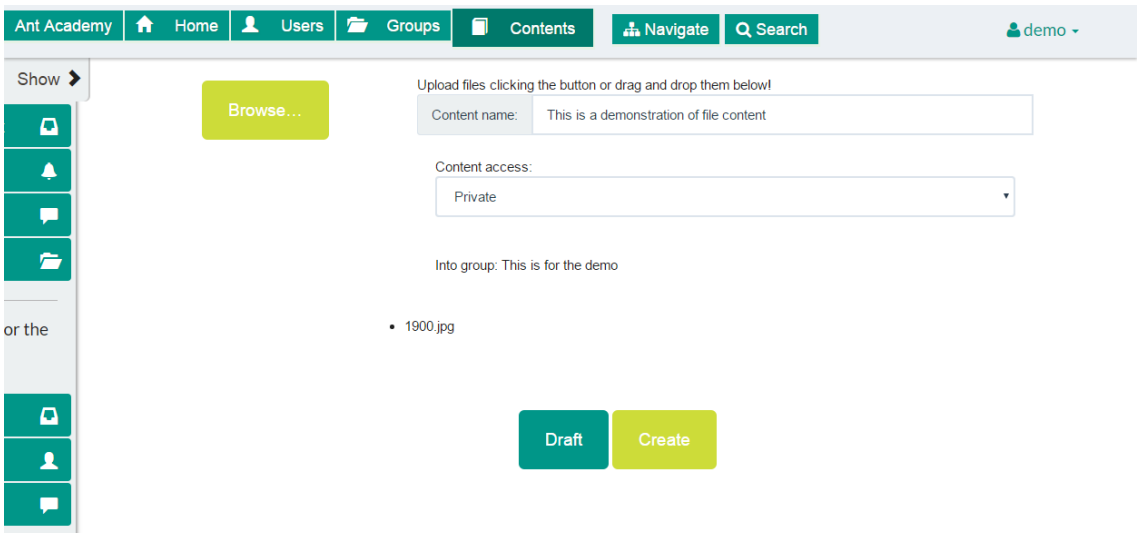


Figure 7.30: File upload inside a group.

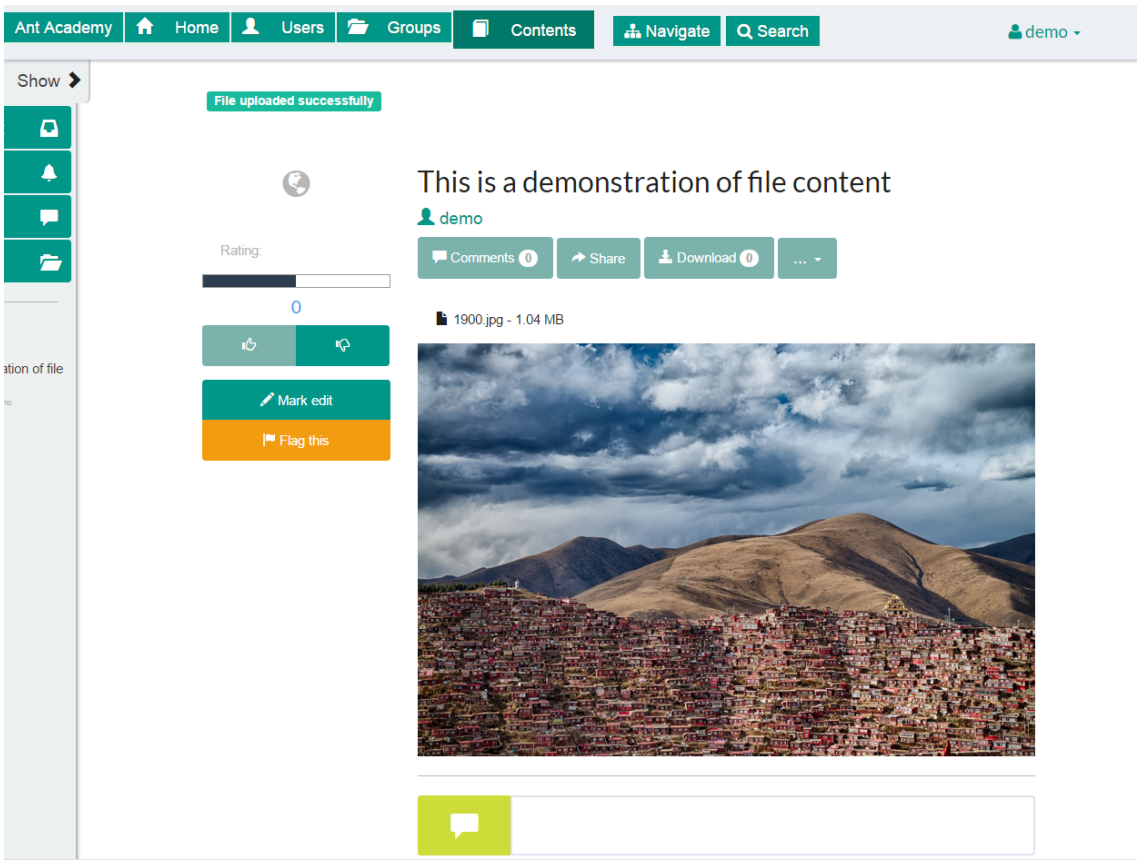


Figure 7.31: Viewing an image file content.

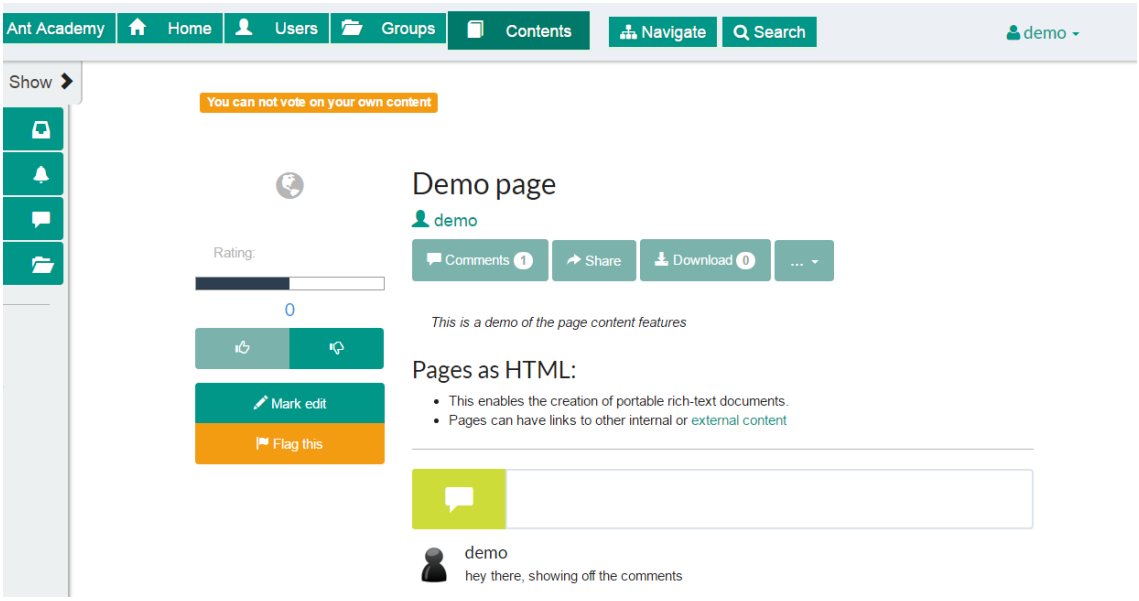


Figure 7.32: Viewing a page content.

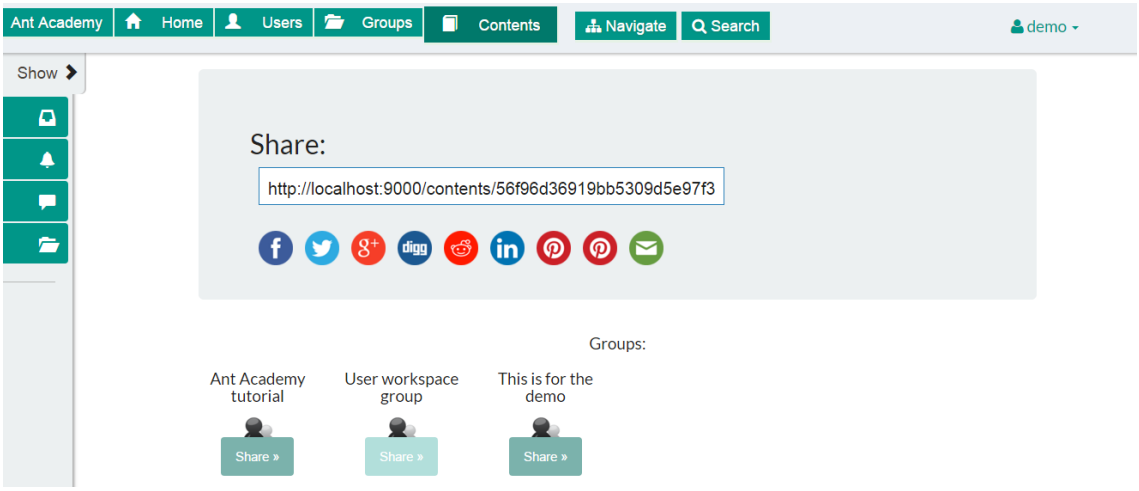


Figure 7.33: Page for sharing the content.

7.11 Administration section

The administration section, shown in Figure 7.34, is only accessible to users with administrative rights on the overall application. This allows to access the list of tags on database, list of requested groups to create (when that option is enabled) and form to invite users.

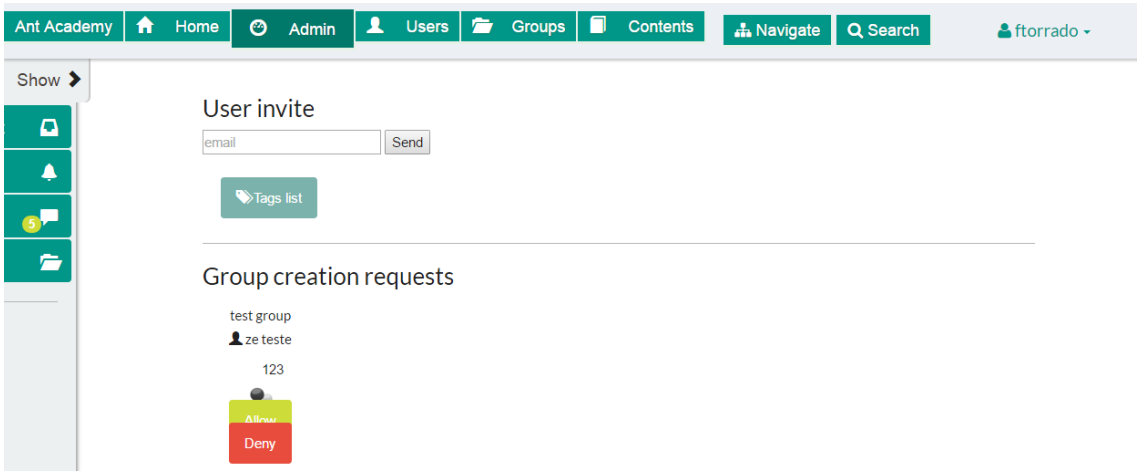


Figure 7.34: Administration section.

7.12 Chat

The users chat system is shown in Figure 7.35, with the first part showing the list of conversations and the SearchHelper tool for selecting users to send messages to. The second part shows the conversation with user "teste1" and its messages.

On the group chat the interface is similar, but there is no need for input to select user or list of conversations.

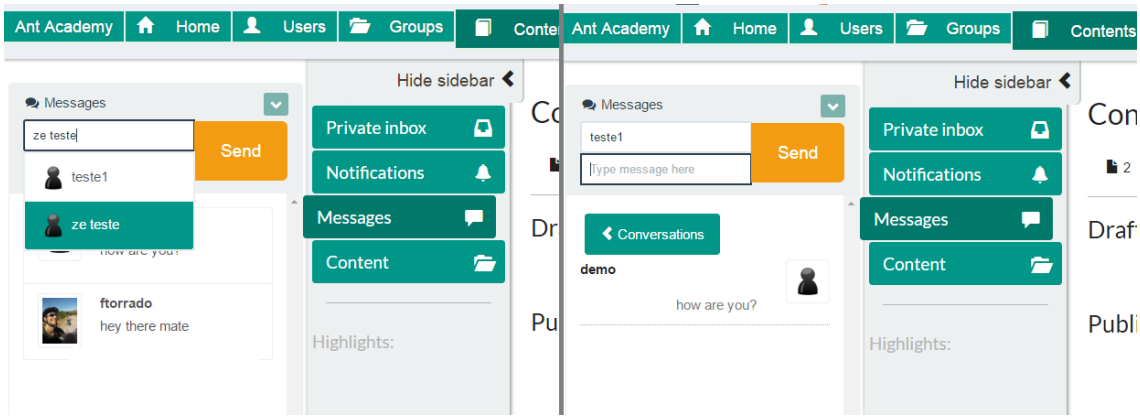


Figure 7.35: User chat showing conversations list and one conversation example.

Chapter 8

Conclusion

As conclusion to this dissertation, a review of the implemented system, results and future work is presented. Surely the project suffered from having goals that were immensely ambitious. Almost trying to develop a full product instead of confining efforts to the development and research of one of the implemented systems: tags and metadata, multi context access control, group based content management, customizable notification feeds for user actions or requests and rating users based on created content quality.

Development goals were satisfactorily achieved, the resulting application could have been more focused on some key aspects, like life cycle of groups and ratings. Yet, the resulting application has a very complete set of implemented features and good presentation. Research goals were partially achieved, lacking real user data and meaningful results. With these results, the dissertation became something more similar to a project implementation report.

This dissertation should have been much less oriented to the development of a system as an appealing application, with superfluous features like asynchronous behaviours and the dynamic sidebar. This led to an extensive period of development, lacking the necessary time to evaluate the system with users and collect data. This would be important for improving the ratings system and to have feedback from users regarding usefulness and usability of the application.

8.1 Lessons Learned

Several lessons learned throughout the development and dissertation.

- Use a Version control system (VCS) like *git* to manage code and other project files, if not already using this is a must.

- Pick a web development framework and language that seem most familiar and easier to understand or read.
- On MVC, the separation between controllers, models and views is hard to get right. Having a good knowledge of the architecture is fundamental before starting development.
- Do not bother much with the looks, implementing features such as the complex sidebar, which is not that useful or good design practice.

8.2 Future Work

Research and development of the subject has many possible directions. In this particular case, the "barebones" implementation requires more features to guarantee interaction between users and to aggregate content from multiple sources or applications.

The rating system should be analysed in a more detailed research, including usage data from the platform in order to simulate and improve the algorithm. It should be also important to make the rating values relevant within groups or group structures, comparing efforts of users within the groups.

On the subject of content, it should have integration with APIs implemented, mainly for file sharing services such as Dropbox and Google Drive. Other feedback elements such as the edit requests and reports that were not fully implemented. As well as enabling multiple users to be authors of the same content.

Groups would benefit much from having a fully functional group structuring system, as well as having multiple group types and pre-sets with more tools than just the groups chat and content sections available. The roles assignment within groups should be more fleshed out, using some kind of voting scheme based on a measurement of trust (probably the rating). Groups should also be allowed to create their own custom group member roles.

The tags based metadata should be used for adding more relevant information to objects, such as related topics, language, etc. This would allow for integration with better user information, relating their interest topics to relevant content and groups.

Bibliography

[1] J. Keengwe and T. T. Kidd. "Towards best practices in online learning and teaching in higher education". In: *Journal of Online Learning and Teaching* 6.2 (2010), p. 533. URL: http://search.proquest.com/openview/61d26c0785c2bcaf470eeef6e52e09b4/1?pq-origsite=gscholar&cbl=2030650 (visited on 03/16/2016).

[2] J. Newmarch. *Lessons from open source: Intellectual property and courseware*. URL: http://firstmonday.org/ojs/index.php/fm/article/view/868/777 (visited on 03/22/2016).

[3] J. Postel, L. G. Roberts, and S. Wolff. "A Brief History of the Internet Barry". In: *Back to Internet Histories* ().

[4] *Brief History of the Internet - Internet Timeline | Internet Society*. URL: http://www.internetsociety.org/internet/what-internet/history-internet/brief-history-internet (visited on 03/12/2016).

[5] *The birth of the web | CERN*. URL: http://home.cern/topics/birth-web (visited on 02/17/2016).

[6] *Standards - W3C*. URL: https://www.w3.org/standards/ (visited on 03/02/2016).

[7] T. Franklin, M. Van Harmelen, and others. "Web 2.0 for content for learning and teaching in higher education". In: *JISC www. jisc. ac. uk/media/documents/programmes/digitalrepositories/web2-contentlearningandteaching. pdf* (2007). URL: https://staff.blog.ui.ac.id/harrybs/files/2008/10/web-2-for-content-for-learning-and-teaching-in-higher-education.pdf (visited on 03/12/2016).

[8] T. O'Reilly. *What Is Web 2.0 - O'Reilly Media*. Sept. 2005. URL: http://www.oreilly.com/pub/a/web2/archive/what-is-web-20.html (visited on 03/12/2016).

[9] *Stack Overflow Developer Survey 2016 Results*. URL: https://stackoverflow.com/research/developer-survey-2016 (visited on 05/23/2016).

[10] L. Parziale, D. T. Britt, C. Davis, J. Forrester, W. Liu, C. Matthews, and N. Rosselot. *TCP/IP Tutorial and Technical Overview*. IBM Redbooks, Dec. 2006. URL: https://www.redbooks.ibm.com/redbooks/pdfs/gg243376.pdf (visited on 03/23/2016).

[11] *RFC 7540 - Hypertext Transfer Protocol Version 2 (HTTP/2)*. URL: https://tools.ietf.org/html/rfc7540 (visited on 03/16/2016).

[12] *Man-in-the-middle attack - OWASP*. URL: https://www.owasp.org/index.php/Man-in-the-middle_attack (visited on 03/22/2016).

[13] *Public Keys and Private Keys - How they work with Encryption | Comodo*. URL: https://www.comodo.com/resources/small-business/digital-certificates2.php (visited on 03/16/2016).

[14] A. S. Tanenbaum and D. Wetherall. *Computer networks*. 5th ed. Boston: Pearson Prentice Hall, 2011. ISBN: 978-0-13-212695-3.

[15] *Understanding The Open Web Stack*. URL: http://blog.smartbear.com/architecture/understanding-the-open-web-stack/ (visited on 03/16/2016).

[16] M. Fowler. *GUI Architectures*. URL: http://martinfowler.com/eaaDev/uiArchs.html (visited on 03/22/2016).

[17] *Web Directions JavaScript, Ajax and the DOM - Web Directions*. URL: http://www.webdirections.org/the-state-of-the-web-2008/javascript-ajax-and-the-dom/ (visited on 03/16/2016).

[18] *Is REST Successful in the Enterprise?* URL: http://www.infoq.com/news/2011/06/Is-REST-Successful (visited on 03/16/2016).

[19] *Web Applications Basics*. URL: http://docs.oracle.com/cd/E13222_01/wls/docs81/webapp/basics.html (visited on 03/29/2016).

[20] K. Matsudaira. *Building Scalable Web Architecture and Distributed Systems | Dr Dobb's*. Dec. 2012. URL: http://www.drdobbs.com/web-development/building-scalable-web-architecture-and-d/240142422 (visited on 03/09/2016).

[21] J. Varia. "Architecting for the cloud: Best practices". In: *Amazon Web Services* (2010). URL: http://www.lifted-llc.com/docs/AWS_Cloud_Architecture_Best_Practices.pdf (visited on 03/29/2016).

[22] *rest/urls · Microformats Wiki*. URL: http://microformats.org/wiki/rest/urls (visited on 03/14/2016).

[23] The Internet Engineering Task Force (IETF). *Main goal of WebSocket protocol*. URL: https://trac.tools.ietf.org/wg/hybi/trac/wiki/FAQ (visited on 03/23/2016).

[24] *PHP: PHP Usage Stats*. URL: http://php.net/usage.php (visited on 03/09/2016).

[25] *Rise of Javascript by Web Development*. URL: https://www.webdigi.co.uk/blog/2009/rise-of-javascript/ (visited on 03/16/2016).

[26] *JSON*. URL: http://json.org/ (visited on 03/08/2016).

[27] *Influential theories of learning | Education | United Nations Educational, Scientific and Cultural Organization*. URL: http://www.unesco.org/new/en/education/themes/strengthening-education-systems/quality-framework/technical-notes/influential-theories-of-learning/ (visited on 03/16/2016).

[28] E. Ackermann. "Piaget's constructivism, Papert's constructionism: What's the difference". In: *Future of learning group publication* 5.3 (2001), p. 438. URL: http://www.sylviastipich.com/wp-content/uploads/2015/04/Coursera-Piaget-_-Papert.pdf (visited on 03/16/2016).

[29] C. Dalsgaard. *Social software: E-learning beyond learning management systems*. URL: http://www.eurodl.org/materials/contrib/2006/Christian_Dalsgaard.htm (visited on 03/29/2016).

[30] *TrainingForce | What is a Learning Management System (LMS)?* URL: http://trainingforce.com/kb/what-is-a-lms/ (visited on 04/21/2016).

[31] J. L. Hennessy. "John L. Hennessy: Risk Taker". In: *IEEE Spectrum, May 2012* (May 2012). URL: http://spectrum.ieee.org/geek-life/profiles/john-l-hennessy-risk-taker.

[32] A. Dasgupta and A. Ghosh. "Crowdsourced judgement elicitation with endogenous proficiency". In: *Proceedings of the 22nd international conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 2013, pp. 319–330. URL: http://dl.acm.org/citation.cfm?id=2488417 (visited on 02/18/2016).

[33] D. A. Muller. "Designing effective multimedia for physics education". PhD thesis. University of Sydney Australia 2008, 2008. URL: http://www.physics.usyd.edu.au/pdfs/research/super/PhD(Muller).pdf (visited on 03/29/2016).

[34] S. Reisman. *The Past, Present and Future of Instructional Technology*. URL: https://www.computer.org/web/computingnow/insights/content?g=53319&type=article&urlTitle=the-past-present-and-future-of-instructional-technology (visited on 04/01/2016).

[35] L. M. Camarinha-Matos, H. Afsarmanesh, N. Galeano, and A. Molina. "Collaborative networked organizations – Concepts and practice in manufacturing enterprises". en. In: *Computers & Industrial Engineering* 57.1 (Aug. 2009), pp. 46–60. ISSN: 03608352. DOI: 10.1016/j.cie.2008.11.024. URL: http://linkinghub.elsevier.com/retrieve/pii/S036083520800301X (visited on 03/11/2016).

[36] R. Soliman, R. Braun, and S. Simoff. "The essential ingredients of collaboration". In: *Collaborative Technologies and Systems, 2005. Proceedings of the 2005 International Symposium on*. IEEE, 2005, pp. 366–373. URL: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1553336 (visited on 03/11/2016).

[37] G. Miao, S. Tao, W. Cheng, R. Moulic, L. E. Moser, D. Lo, and X. Yan. "Understanding task-driven information flow in collaborative networks". In: *Proceedings of the 21st international conference on World Wide Web*. ACM, 2012, pp. 849–858. URL: http://dl.acm.org/citation.cfm?id=2187951 (visited on 03/11/2016).

[38] Z. Du, X. Fu, C. Zhao, Q. Liu, and T. Liu. "Interactive and Collaborative E-Learning Platform with Integrated Social Software and Learning Management System". en. In: *Proceedings of the 2012 International Conference on Information Technology and Software Engineering*. Ed. by W. Lu, G. Cai, W. Liu, and W. Xing. Lecture Notes in Electrical Engineering 212. DOI: 10.1007/978-3-642-34531-9_2. Springer Berlin Heidelberg, 2013, pp. 11–18. ISBN: 978-3-642-34530-2 978-3-642-34531-9. URL: http://link.springer.com/chapter/10.1007/978-3-642-34531-9_2 (visited on 03/22/2016).

[39] G. Servin and C. De Brun. "ABC of knowledge management". In: *NHS National Library for Health: Specialist Library* (2005). URL: http://www.fao.org/fileadmin/user_upload/knowledge/docs/ABC_of_KM.pdf (visited on 03/29/2016).

[40] W.-H. Wu, Y.-C. Jim Wu, C.-Y. Chen, H.-Y. Kao, C.-H. Lin, and S.-H. Huang. "Review of trends from mobile learning studies: A meta-analysis". en. In: *Computers & Education* 59.2 (Sept. 2012), pp. 817–827. ISSN: 03601315. DOI: 10.1016/j.compedu.2012.03.016. URL: http://linkinghub.elsevier.com/retrieve/pii/S0360131512000735 (visited on 03/29/2016).

[41] Z. Liang and W. Shi. "Analysis of ratings on trust inference in open environments". en. In: *Performance Evaluation* 65.2 (Feb. 2008), pp. 99–128. ISSN: 01665316. DOI: 10.1016/j.peva.2007.04.001. URL: http://linkinghub.elsevier.com/retrieve/pii/S0166531607000430 (visited on 02/18/2016).

[42] A. Anderson, D. Huttenlocher, J. Kleinberg, and J. Leskovec. "Steering user behavior with badges". In: *Proceedings of the 22nd international conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 2013, pp. 95–106. URL: http://dl.acm.org/citation.cfm?id=2488398 (visited on 02/18/2016).

[43] *How Reddit ranking algorithms work — Hacking and Gonzo — Medium*. URL: https://medium.com/hacking-and-gonzo/how-reddit-ranking-algorithms-work-ef111e33d0d9 (visited on 03/24/2016).

[44] *How Hacker News ranking algorithm works — Hacking and Gonzo — Medium*. URL: https://medium.com/hacking-and-gonzo/how-hacker-news-ranking-algorithm-works-1d9b0cf2c08d#.19o3e8nkn (visited on 03/24/2016).

[45] Z. Liang and W. Shi. "Performance evaluation of rating aggregation algorithms in reputation systems". In: *Collaborative Computing: Networking, Applications and Worksharing, 2005 International Conference on*. IEEE, 2005, 10–pp. URL: http:

//ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1651235 (visited on 02/18/2016).

[46] A. Abdel-Hafez, Y. Xu, and A. Jøsang. "A normal-distribution based reputation model". In: *Trust, Privacy, and Security in Digital Business*. Springer, 2014, pp. 144–155. URL: http://link.springer.com/chapter/10.1007/978-3-319-09770-1_13 (visited on 03/21/2016).

[47] *Play Frameork - The MVC application model*. URL: https://www.playframework.com/documentation/1.0/main (visited on 03/01/2016).

[48] *passlib.hash.bcrypt - BCrypt — Passlib v1.6.5 Documentation*. URL: http://pythonhosted.org/passlib/lib/passlib.hash.bcrypt.html (visited on 03/19/2016).

[49] *SHA-1 Broken - Schneier on Security*. URL: https://www.schneier.com/blog/archives/2005/02/sha1_broken.html (visited on 03/19/2016).

[50] *Transition from SHA-1 to SHA-2 Certificates | Symantec*. URL: http://www.symantec.com/page.jsp?id=sha2-transition (visited on 03/19/2016).

[51] *ObjectId — MongoDB Manual 3.2*. URL: https://docs.mongodb.org/manual/reference/object-id/ (visited on 03/02/2016).

[52] *Reference — MongoDB Manual 3.2*. URL: https://docs.mongodb.org/manual/reference/ (visited on 03/02/2016).

[53] R. S. Sandhuy. "Role-based access control". In: (1997). URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.26.2311&rep=rep1&type=pdf (visited on 03/24/2016).

Appendix A

Project Routes

The full list of URLs implemented on the platform was copied from the routes configuration file used on the platform, with a total of 104 routes. Higher priority routes come first on the list.

Home page and landing pages

Listing A.1: Main routes

```
1 GET   /                   controllers.Application.welcome()
2 GET   /about              controllers.Application.about()
3 GET   /contact            controllers.Application.contact()
4 GET   /old-sketches       controllers.Application.oldSketches()
5 GET   /notify/:section    controllers.Application.showEmptyPage(filler ?= null
      , section, goBack ?=null)
6 GET   /feedback           controllers.Application.feedback()
7 POST  /subscribe          controllers.Application.subscribe()
8 GET   /app                controllers.Application.index()
```

Administration

Listing A.2: Administration routes

```
1 GET   /admin              controllers.Admin.index()
2 GET   /admin/:accessor    controllers.Admin.backOffice(accessor)
3 POST  /admin/:accessor    controllers.Admin.backOfficeSetup(accessor)
4 POST  /admin/invite       controllers.Admin.inviteUser()
5 POST  /admin/groups/:groupId    controllers.Admin.manageGroup(groupId)
```

Chat messages

Listing A.3: Chat messages routes

```
1  GET   /messages           controllers.ChatMessages.index()
2  GET   /messages/:user     controllers.ChatMessages.withUser(user)
3  POST  /messages           controllers.ChatMessages.postMessage()
4  GET   /groups/:id/chat    controllers.ChatMessages.groupChat(id, isSidebar:
       java.lang.Boolean ?= java.lang.Boolean.TRUE)
```

Actions

Listing A.4: Actions routes

```
1  GET   /actions/:actionId          controllers.Actions.showAction(actionId)
2  POST  /actions/:actionId/accept   controllers.Actions.acceptRequest(actionId)
3  POST  /actions/:actionId/reject   controllers.Actions.rejectRequest(actionId)
4  GET   /users/:user/actions        controllers.Actions.userActions(user)
5  GET   /groups/:group/actions      controllers.Actions.groupActions(group)
6  GET   /groups/:group/members/:member/actions    controllers.Actions.
       groupUserDigest(group, member)
7  GET   /contents/:content/actions    controllers.Actions.contentActions(content
       )
```

Users pages and profiles

Listing A.5: Users routes

```
1  GET   /users              controllers.Users.index()
2  GET   /users/search       controllers.Users.search()
3  GET   /users/me           controllers.Users.showMyProfile()
4  GET   /users/me/edit      controllers.Users.editMyProfile()
5  POST  /users/me/profile-pic   controllers.Users.changeProfilePicture()
6  GET   /users/:user        controllers.Users.showProfile(user)
7  GET   /users/:user/groups     controllers.Users.listGroups(user)
8  GET   /users/:user/contents   controllers.Users.listContents(user)
9  GET   /users/:user/img    controllers.Users.getThumbnail(user)
10 POST  /users/me/img       controllers.Users.changeProfilePicture()
11 POST  /users/search-helper    controllers.Users.searchHelper()
```

Groups

Listing A.6: Groups routes

```
1  GET   /groups             controllers.Groups.index()
2  POST  /groups             controllers.Groups.create()
3  GET   /groups/create      controllers.Groups.creationForm()
4  GET   /groups/create/:id    controllers.Groups.creationFormSubgroup(id)
5  GET   /groups/search      controllers.Groups.search()
6  GET   /groups/navigator   controllers.Groups.navigator()
7  GET   /groups/workspace   controllers.Groups.privateWorkspace()
8  GET   /groups/:id/inbox   controllers.Groups.showInbox(id)
9  POST  /groups/:id/inbox   controllers.Groups.addToInbox(id)
10 POST  /groups/:id/inbox/:request      controllers.Groups.inboxResponse(id,
       request)
11 GET   /groups/:id         controllers.Groups.show(id)
12 POST  /groups/:id/delete    controllers.Groups.delete(id)
13 GET   /groups/:id/delete    controllers.Groups.delete(id)
14 POST  /groups/:id/members   controllers.Groups.memberManage(id)
15 POST  /groups/:id/members/invite      controllers.Groups.inviteUser(id)
16 POST  /groups/:id/members/requests/   controllers.Groups.requestMembership(id)
17 POST  /groups/:id/members/requests/:user    controllers.Groups.
       membershipRequestResponse(id,user)
18 DELETE    /groups/:id/members/:member   controllers.Groups.removeMember(id,
       member)
19 POST  /groups/:id/members/:member/tiers/:tier   controllers.Groups.
       promoteMemberTier(id,member,tier)
20 GET   /groups/:id/img     controllers.Groups.getThumbnail(id)
21
22 POST  /groups/:id/settings    controllers.Groups.changeGroupSettings(id)
23 GET   /groups/:id/settings/permissions    controllers.Groups.
       settingsPermissions(id)
24 POST  /groups/:id/settings/permissions    controllers.Groups.runPermissions(id
       )
25 GET   /groups/:id/settings    controllers.Groups.showSettings(id)
26 GET   /groups/members/:id     controllers.Groups.showMember(id)
```

Group structures

Listing A.7: Group structures routes

```
1  GET   /groups/:id/structs          controllers.GroupStructs.showGroupStruct(
       id)
2  POST  /groups/:id/structs/parent    controllers.GroupStructs.structureParent(
       id)
3  POST  /groups/:id/structs/child     controllers.GroupStructs.structureChild(id
       )
```

Content

Listing A.8: Content routes

```
1  GET   /contents           controllers.Contents.index()
2  POST  /contents           controllers.Contents.create(groupId = null, ctype)
3  POST  /groups/:groupId/contents   controllers.Contents.create(groupId, ctype)
4  GET   /contents/search    controllers.Contents.search()
5  GET   /contents/navigator   controllers.Contents.navigator()
6  GET   /contents/editor    controllers.Contents.creationForm(groupId = null,
       ctype, id = null)
7  GET   /contents/editor/:id    controllers.Contents.creationForm(groupId = null
       , ctype ?= null, id)
8  GET   /groups/:groupId/contents/editor    controllers.Contents.creationForm(
       groupId, ctype ?= null, id = null)
9  GET   /groups/:groupId/contents/editor/:id    controllers.Contents.
       creationForm(groupId, ctype ?= null, id)
10 GET   /contents/:id       controllers.Contents.show(id, ver: Integer ?= null)
11 PUT   /contents/:id       controllers.Contents.update(id)
12 POST  /contents/:id/delete      controllers.Contents.delete(id)
13 GET   /contents/:id/delete      controllers.Contents.delete(id)
14 GET   /contents/:id/download    controllers.Contents.download(id)
15 GET   /contents/:id/share       controllers.Contents.shareMenu(id)
16 POST  /contents/:id/comment     controllers.Contents.comment(id)
17 POST  /contents/:id/vote        controllers.Contents.vote(id)
18 POST  /contents/:id/pin   controllers.Contents.pin(id)
19 GET   /contents/:id/groups      controllers.Contents.showGroups(id)
20 POST  /contents/:id/groups/:groupId   controllers.Contents.shareToGroup(id,
       groupId,parentGroupId ?= null)
21 GET   /contents/:id/img   controllers.Contents.getThumbnail(id)
```

Tags

Listing A.9: Tags routes

```
1  GET   /tags               controllers.Tags.index()
2  GET   /tags/:ns           controllers.Tags.showNamespace(ns)
3  GET   /tags/:ns/:pr       controllers.Tags.showTag(ns, pr)
```

User settings

Listing A.10: User settings routes

```
GET   /settings            controllers.account.Settings.index()
GET   /settings/privacy    controllers.account.Settings.showPrivacy()
POST  /settings/privacy    controllers.account.Settings.runPrivacy()
GET   /settings/password     controllers.account.Settings.showPassword(token)
POST  /settings/password     controllers.account.Settings.runPassword(token)
GET   /settings/password/lost   controllers.account.Settings.showPasswordLost
    ()
POST  /settings/password/lost   controllers.account.Settings.runPasswordLost()
GET   /settings/email      controllers.account.Settings.showEmail()
POST  /settings/email      controllers.account.Settings.runEmail()
```

User session

Listing A.11: User session routes

```
GET   /login               controllers.Application.login()
POST  /login               controllers.Application.authenticate()
GET   /logout              controllers.Application.logout()
GET   /signup              controllers.account.Signup.create()
POST  /signup              controllers.account.Signup.save()
POST  /signup/:token/accept     controllers.account.Signup.acceptInvite(token)
POST  /signup/:token/reject     controllers.account.Signup.rejectInvite(token)
GET   /confirm/:confirmToken    controllers.account.Signup.confirm(
    confirmToken:String)
```

Header, sidebar and navigation

Listing A.12: Site navigation routes

```
GET   /views/header        controllers.Utils.showHeader(section: String ?= "")
GET   /views/sidebar       controllers.Utils.showSidebar()
GET   /views/sidebar/group/:id      controllers.Utils.showGroupSidebar(id:
    String)
GET   /views/sidebar/content/:id    controllers.Utils.showContentSidebar(id:
    String)
GET   /views/jsmessages    controllers.Utils.javascriptMessages()
GET   /views/navigator     controllers.Utils.navigator()
GET   /views/search        controllers.Utils.search()
```

Other

Listing A.13: Other routes

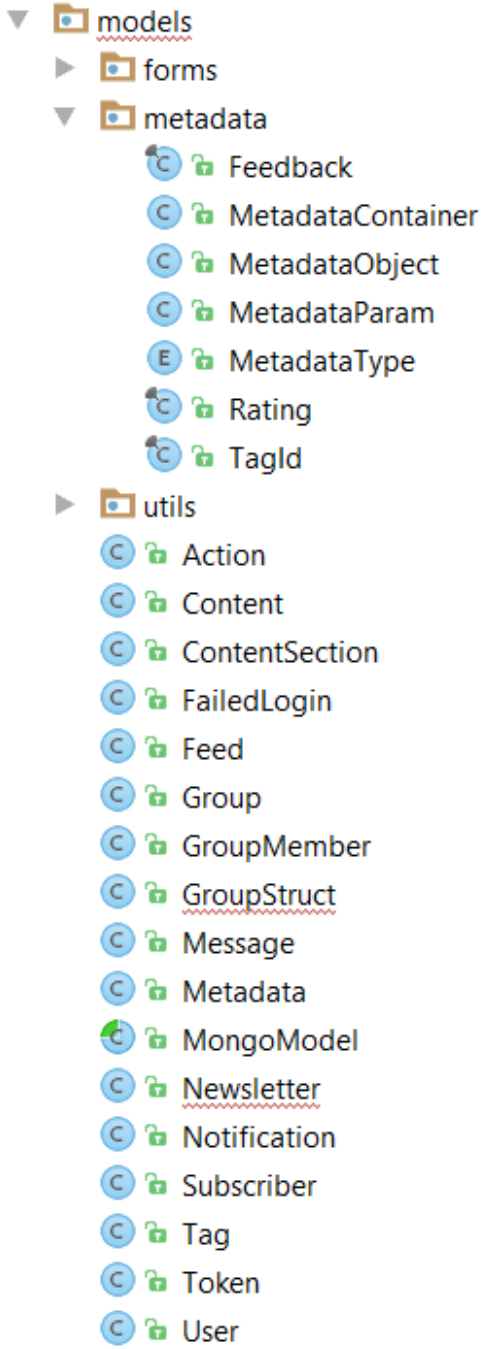
```
1 GET   /apis/google/drive/oauth2-callback    controllers.apis.GoogleDrive.
      callback(redirect : String ?= null)
2 GET   /assets/*file       controllers.Assets.at(path="/public", file)
```

Appendix B

Project File Structure

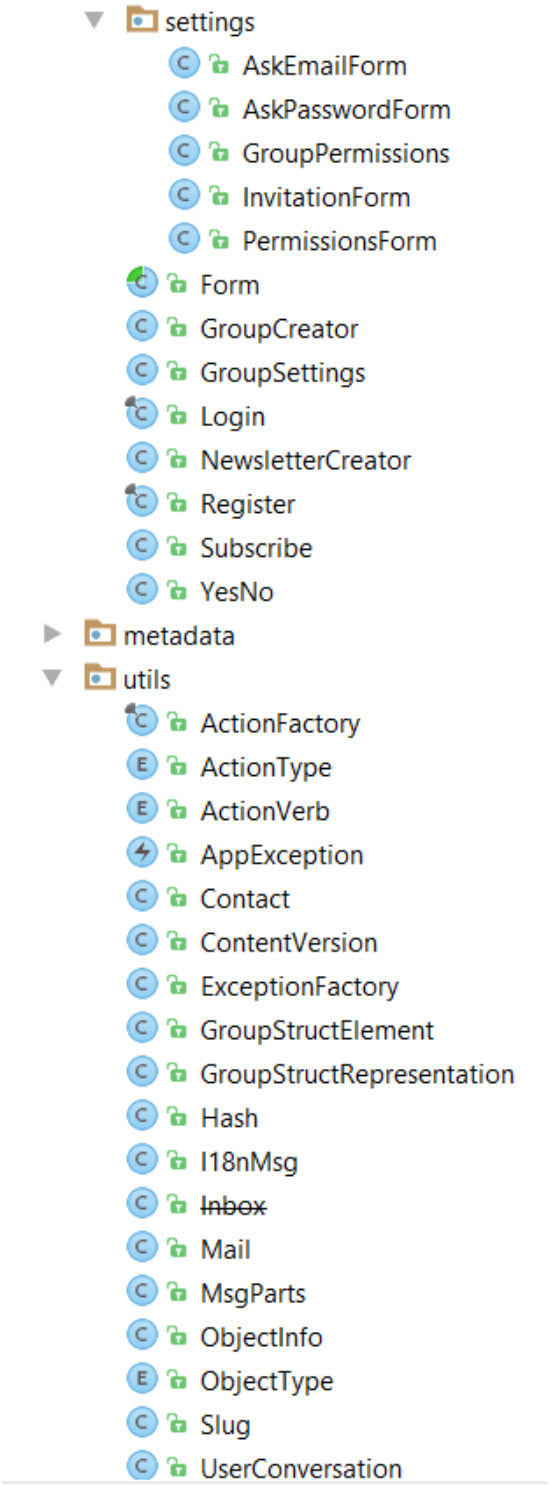
The images on this appendix show the list of the project code and main configuration files, totalling 204 source code files, without taking into account external libraries used in the project such as Bootstrap and jQuery files.

(a) Controllers

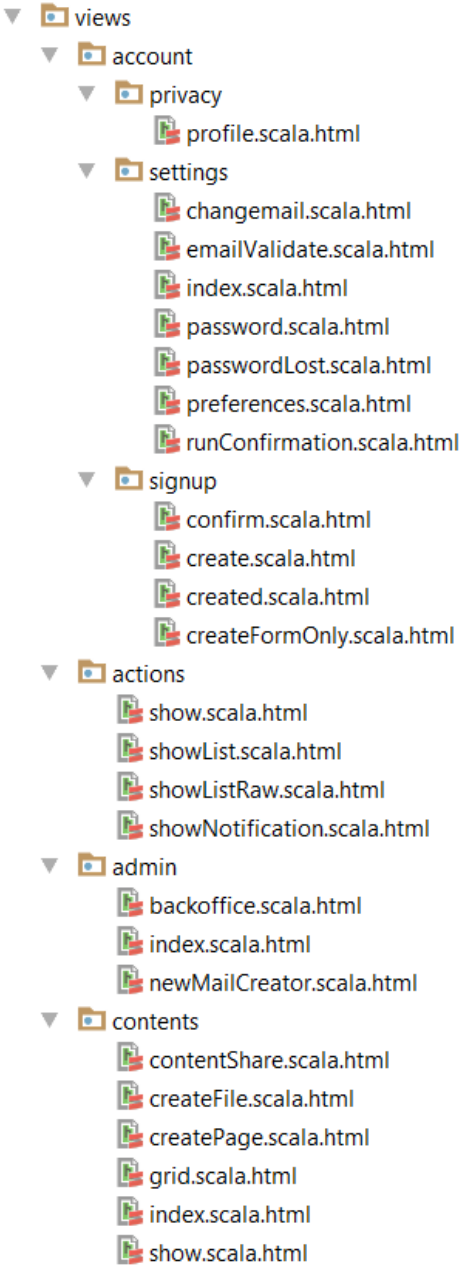


(b) Models, part 1 of 2

Figure B.1: Controllers and models (1 of 2) files

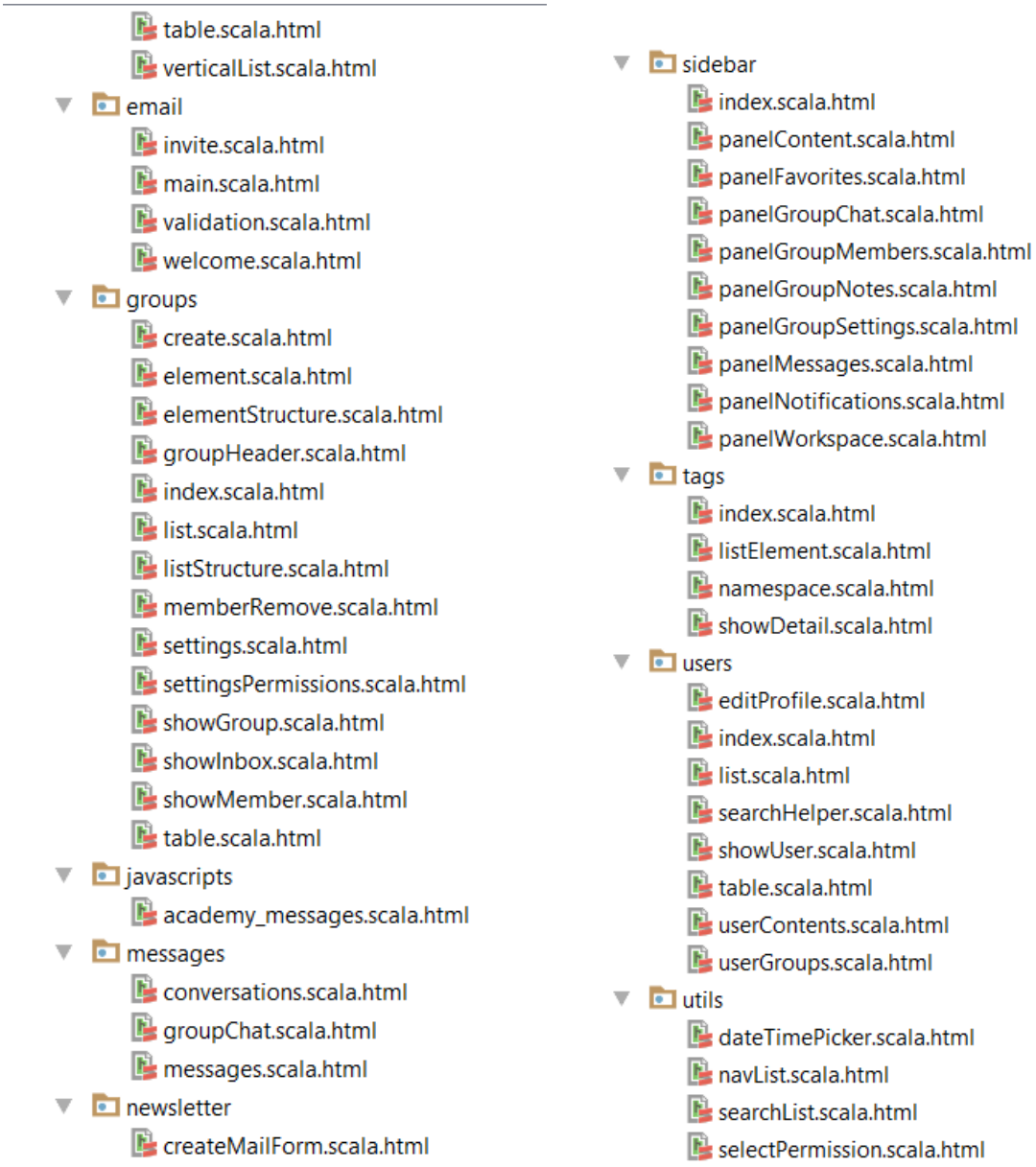


(a) Models, part 2 of 2



(b) Views, part 1 of 4

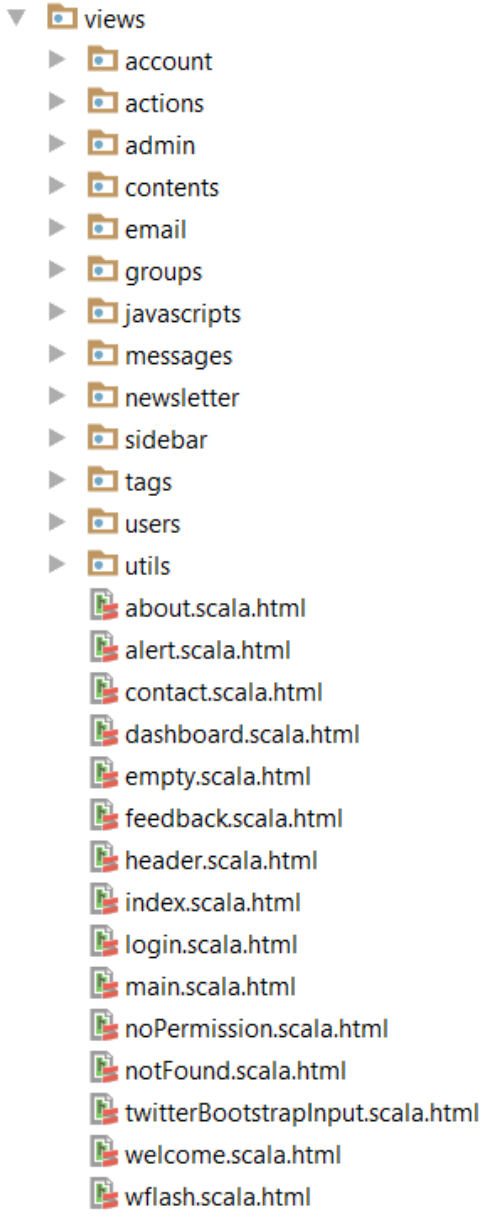
Figure B.2: Models (2 of 2) and views (1 of 4) files



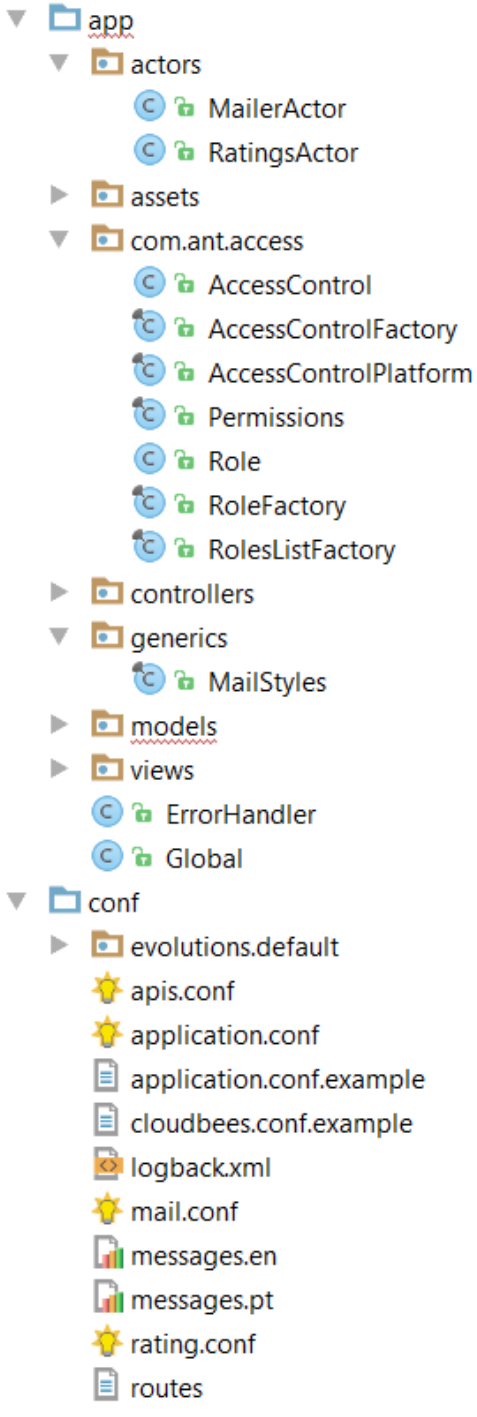
(a) Views, part 2 of 4

(b) Views, part 3 of 4

Figure B.3: Views (2 and 3 of 4) files

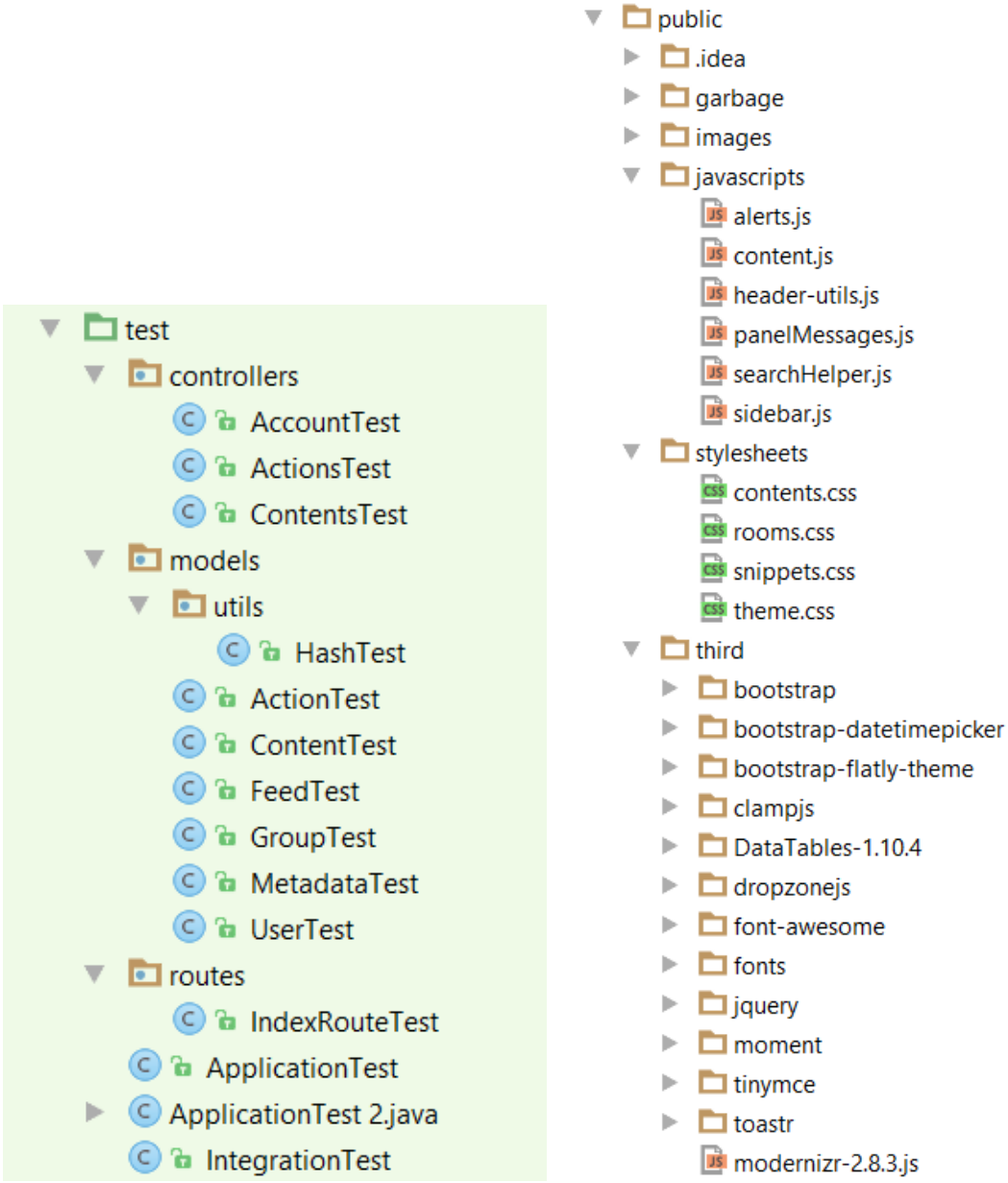


(a) Views, part 4 of 4



(b) Access control, utilities and configuration

Figure B.4: Views (4 of 4), access control, utilities and framework configuration files



(a) Tests

(b) JavaScript, CSS and assets

Figure B.5: Unit tests, JavaScript, CSS and front-end asset files